SYSTEMS
DESIGN
Handbook

FIRST EDITION*

Monolithic Memories

* Additional Application Note

Introduction

Monolithic Memories has been providing innovative solutions to complex digital system design problems since its inception in 1969. High-speed PROM, PAL® and HAL® integrated circuits, FIFO memories, fully parallel 8- and 16-bit multipliers, and Double-Density Interface™ are among the industry firsts provided by Monolithic Memories for digital system designers.

The requirements of modern systems demand components with increased capabilities. Rapid progress in memory devices demands new levels of performance in memory support chips, like our Dynamic RAM Controllers and Drivers. Unprecedented interest in reliability makes fault-tolerant and error-detection and -correction schemes part of many modern digital systems. Diagnosing and correcting system problems requires new components with on-chip diagnostic capabilities, like our Diagnostic PROMs and Registered PROMs. High-speed computing and the proliferation of digital techniques for signal processing is made possible by new components like our high-speed parallel multipliers and serial multiplier/dividers. Efficient communications between digital systems and subsystems is facilitated by use of members of our family of FIFO memories, and the requirements of bringing more compact systems to market with shorter design cycles are served by our PAL and PLE™ families of user-programmable logic devices and their supporting CAD tools, PALASM and PLEASM.

The challenges of advanced system design require innovative architecture as well as innovative LSI building blocks. This book is a collection of application notes pertaining to a variety of system design topics. They contain unique approaches to solving both large and small system design problems. They are organized by system type to provide you, the system designer, with a useful reference for digital systems design.

Table of Contents

Digital Signal Processing

Fault-Tolerant Systems

Computer Design

Digital Arithmetic

Industrial Control

Memory Support

FIFO Buffer Design

Interface Techniques

Communication Systems

Monolithic Memories MMI

Index by Product Family

PROM

PAL

FIFO

Memory Support Series

Arithmetic Elements and Logic

Multiplier/Dividers

Interface

PROMs, PALs, FIFOs, and Multipliers. Team up to Implement Single-Board High-Performance Audio Spectrum Analyzer

Richard Wm. Blasco

A high-performance audio signal analyzer can be built using specialized arithmetic logic, first-in-first-out memories (FIFOs), programmable-array-logic devices (PALs), and PROMs. The specialized arithmetic logic consists of the versatile 74S516 16-bit multiplier/divider/multiplier/ accumulator.

Digital Signal Processing (DSP) is a realizable alternative to analog techniques. However, single-chip DSP devices are sometimes very limited in performance. Another solution, more expensive in dollars and board real estate, is the bit-slice approach. Customized architecture, which is desirable, can best be achieved with the methodology and design described in the application note; a designer has very limited flexibility using either the bit-slice or the single-chip approach.

The design technologies used to implement the analyzer are quite general and can be applied to a variety of DSP tasks. An understanding of the approach used here should suggest solutions to a number of other DSP problems, which can in some cases be implemented by altering the micro-program which controls the spectrum analyzer without hardware chanages. The high-performance architecture using PROMs, FIFOs, multipliers, and PALs therefore provides an optimum price/performance alternative to other DSP approaches.

PROMS, PALS, FIFOS, and Multipliers
Team up to Implement Single-Board High-Performance Audio Spectrum Analyzer

Richard Wm. Blasco

Introduction

Digital signal processing (DSP) is rapidly becoming a popular alternative to analog techniques in high-performance applications. However, single-chip DSP devices provide only limited capability. Bit-slice processors can improve performance, but occupy excessive board space. Improved efficiency can be achieved with customized architectures, but both the single-chip and bit-slice approaches impose limitations on the designer's flexibility.

Now programmable array logic (PAL) devices, PROMs, first in-first out buffers (FIFOs), and multiplier chips can team up to implement customized architectures while keeping total package count equivalent to bit-slice solutions. The PAL devices, packaged in 20-pin skinny-DIPs, occupy less board space than 40-pin bit-slice devices. Because the PAL functions are customized by the user, PALs can provide DSP power equivalent to the bit-slices, even though the functional density is slightly less. Users get only the functions they need, and save board space as a result.

This approach is illustrated by a high-performance audio spectrum analyzer. This circuit can analyze high-fidelity audio signals with a resolution of 20 Hz and an input bandwidth of 20-kHz. It is useful in production test, performance evaluation, or adjustment of high-fidelity audio equipment. The analyzer provides a swept generator output for rapid analysis of audio filter frequency response.

The design techniques used to implement the analyzer are quite general, and can be applied to a wide variety of DSP tasks. An understanding of the approach used will suggest solutions to a number of DSP problems. The architecture chosen for the spectrum analyzer is controlled by a microprogram stored in PROM. Many other applications can be accommodated by changing the microprogram. The high performance of this architecture provides an attractive price/performance alternative to other DSP approaches.

Spectrum Analyzer Functions

The spectrum analyzer requires many of the functions commonly used in DSP. Fig. 1 shows the analyzer functions. An input signal is mixed with a swept audio sinewave oscillator. Mixing is accomplished by multiplying the input signal by the sinewave. From basic trigonometry,

$$\cos w_1 t \times \cos w_2 t = \tfrac{1}{2} \cos (w_1 + w_2)t + \tfrac{1}{2} \cos (w_1 - w_2)\, t \quad (1)$$

The mixing process generates two new sinewaves whose frequencies are the sum and difference of the input sinewave frequencies. When the sinewave oscillator matches the frequency of an input signal component, a DC term is generated in proportion to the amplitude of that component.

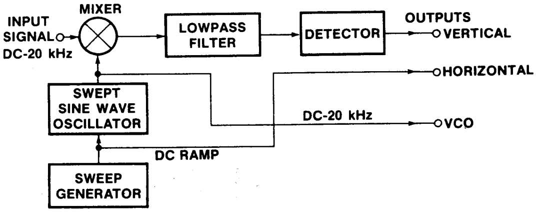


Figure 1. Spectrum Analyzer Functions

The DC term is extracted by a narrow lowpass filter. Due to the finite bandwidth of this lowpass filter, output signals whose frequencies fall within the filter passband also appear at the filter output. As a result, the analyzer output will represent the energy contained in a range of frequencies, from the sinewave frequency minus the filter cutoff frequency, to the sinewave frequency plus the filter cutoff frequency. The effective bandwidth of the analyzer is twice the lowpass filter bandwidth.

A detector converts the lowpass filter output to a DC voltage representing the total energy in the filter passband. If this DC voltage is plotted on a vertical axis with the sinewave oscillator frequency (represented by the sweep voltage) controlling the horizontal axis, the spectrum of the input signal results.

Other mixing schemes can be used to extract the spectrum. However, this "direct conversion" approach has two significant advantages. As shown in Fig. 2, the swept oscillator output can be used to plot the frequency response of an audio filter. Other schemes require additional mixing to achieve the same result.

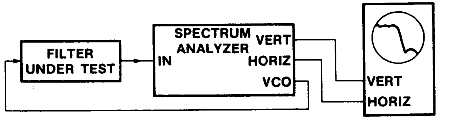


Figure 2. Filter Test Mode Setup

The direct conversion scheme confines the frequencies of all signals following the mixer to the lowpass filter bandwidth. Limiting the signal bandwidth has great benefit when the analyzer is implemented digitally. This benefit can be better understood with a brief review of DSP theory.

Digital Signal Processing Theory Review

Digital signal processing is accomplished by first converting the continuous analog input signal to a series of digital numbers. The digital numbers are then manipulated to perform the required signal processing. The processed digital numbers are converted back to a continuous analog signal, completing the processing. The functions required for DSP are shown in Fig. 3.

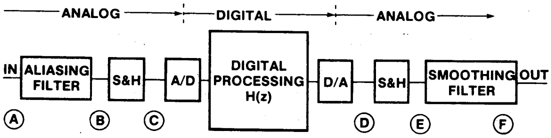


Figure 3. DSP Functions

Sampling

Representing a continuous input signal would require an infinite array of digital numbers. A finite collection of digital numbers can be obtained by considering the signal amplitude at discrete, periodic points in time. This process is called **sampling,** and is equivalent to multiplying the input signal by a periodic train of impulses of unit amplitude. The **sampling theorem** states that the input signal can be reconstructed without distortion if the input is bandlimited to contain no frequency components greater than half the sampling frequency. The sampling theorem means that the discrete samples completely represent the input signal, as long as the bandwidth constraint is met.

Aliasing

What is really happening during the sampling process? Consider the Fourier series representation of a periodic unit impulse train. It can be shown that:

$$f(t) = \sum_{k=-\infty}^{k=\infty} \cos(2\pi k f_s t),\ k = 0, 1, 2, 3, \ldots \qquad (2)$$

$$\text{where } f_s = \frac{1}{\text{sample period}}$$

The periodic impulse train is equivalent to a series of sinusoids consisting of all harmonics of the sampling frequency, including a DC term. Recalling Eq. (1), all possible sum and difference frequencies will be generated when the impulse train and the input signal are multiplied. This process is shown graphically in Fig. 4. Observe that if the input contains frequencies greater than half the sampling frequency, the spectra in Fig. 4 will overlap. This overlap phenomenon is known as **aliasing distortion,** and introduces noise in the signal.

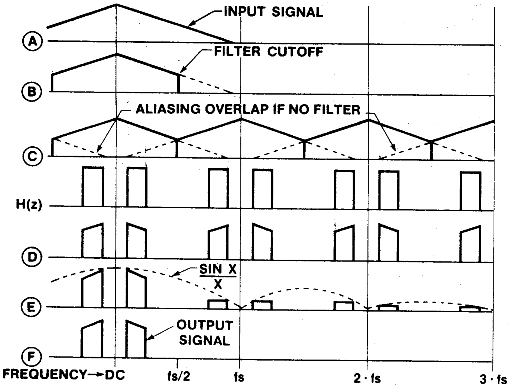


Figure 4. Aliasing Spectra of Figure 3 DSP Functions

Another consequence of the sampling process is that high-frequency signal components near a harmonic of the sampling frequency will be mixed to produce new signal components near DC. These new components have frequencies within the desired signal passband, but are really "alias" high-frequency components. The phenomenon is called **aliasing.**

To eliminate the undesirable effects of aliasing, a continuous analog lowpass filter is placed before the sampler. This **aliasing filter** removes frequency components beyond the $f_s/2$ limit.

Quantizing

The input samples are converted to a series of digital numbers by an analog-to-digital (A/D) converter. The A/D converter operates by **quantizing** the continuous sample amplitude into a finite number of amplitude ranges, and then assigning a digital number to represent the quantized amplitude value. As might be expected, this process introduces noise in the signal, known as **quantization distortion.** The quantization distortion is in the form of a "white" or broadband random noise, whose RMS amplitude is:

$$\sigma^2 = \frac{1}{12}2^{-2b} \qquad (3)$$

where b is the number of bits in the output digital word, excluding the sign bit

The effect of aliasing on quantization noise is to alias high frequency noise components to the DC to $f_s/2$ range. The resulting noise spectral density is equivalent to a white noise of amplitude σ^2, bandlimited to $f_s/2$.

Dynamic Range

The A/D output contains a finite number of bits. **Dynamic range** is defined as the ratio of the maximum to minimum signal amplitude that can be represented by the digital numbers. Dynamic range is determined by the number of bits in the digital numbers, and by the noise "floor."

For a digital number containing b bits plus a sign bit, the dynamic range would be:

$$\text{Dynamic range (dB)} = 10 \log_{10} 2^{-2b} \qquad (4)$$

The noise floor is the sum of all noise components that can appear at the DSP output. The primary noise factors are quantization noise and limit cycle noise (to be discussed shortly). Digital filtering will affect the noise floor by eliminating components of the noise signal. For example, the quantization noise at the DSP output is:

$$N_Q\ \text{(dB)} = 10 \log_{10} \left[\sigma^2 \frac{BW}{f_s/2}\right] \qquad (5)$$

where BW is the net bandwidth of the digital filters

The noise components are uncorrelated, and are therefore combined by adding the power of each noise component. Remember that

$$\text{Power (absolute)} = \log_{10}^{-1} \left[\text{Power (dB)}/10\right] \qquad (6)$$

The resulting dynamic range is:

$$\text{Dynamic range (dB)} = 10 \log_{10} \frac{0.5}{\Sigma \text{ Noise Power}} \qquad (7)$$

where 0.5 = the maximum mean-squared amplitude

The overall dynamic range is the lesser of the result given by Eq. (4) or Eq. (7). In a practical system, the width of the digital numbers can vary. The dynamic range is usually calculated for all critical points in a digital system, with the overall dynamic range being the worst case value.

Digital Processing

The digital numbers from the A/D converter are manipulated to process the signal. Carrier generation, filtering, and nonlinear operations are performed by appropriate "number crunching".

Generation of sinusoidal carriers is easily accomplished using a linear ramp function (digital up/down counter) and converting the results to sinusoidal samples using ROM lookup tables. Alternately, recursive equations can produce the desired carriers.

Nonlinear operations on the digital numbers must be handled with care. Since aliasing is always present in the sampled domain, harmonics generated by nonlinear operations can alias to lower frequencies. The aliasing occurs "immediately," since it can be shown that performing a nonlinear operation in the sampled domain is equivalent to first performing the nonlinear operation on a continuous signal and then sampling the result without bandlimiting the sampler input.

The sampling rate can be changed to improve the efficiency of the digital processing. For example, discarding every other digital number would reduce the effective sampling rate by a factor of two. If the processing at the higher sample rate includes digital aliasing filters to remove components greater than half the lower sample rate, the requirements of the sampling theory are still met. The sampling rate can be increased by repeating digital sample values. This repetition is equivalent to a "sample and hold" operation, and modifies the signal spectrum by

$$F'(jw) = F(jw) \times \frac{\sin (wT/2)}{wT/2} \qquad (8)$$

where $w = 2\pi \times$ freq

T = input (longer) sample period

The effects of changing the sampling rate are best determined by plotting the resulting aliasing spectra.

Digital Filtering

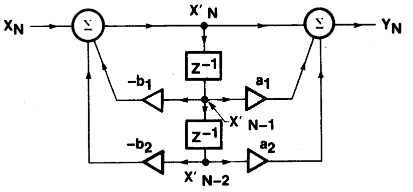
Digital filtering is accomplished using multiplication, addition, and delay. For example, consider the biquadradic filter section in Fig. 5. If z^{-1} is defined to be a unit sample period delay operator, then the input-to-output transfer function of the biquadratic section is:

$$H(z) = \frac{1 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \qquad (9)$$

The biquadric sections can be cascaded to implement higher-order filters.

The Laplace transform of a unit delay is e^{-sT}, where T is the delay period. Remember that z^{-1} represents an inverse operator, so that $z \times z^{-1} = 1$. Thus,

$$z = e^{sT}, \text{ where } s = \alpha + jw \qquad (10)$$



$$X'_N = X_N - b_1 X'_{N-1} - b_2 X'_{N-2}$$
$$Y_N = X'_N + a_1 X'_{N-1} + a_2 X'_{N-2}$$

Figure 5. Digital Biquadratic Filter Section

Digital filter poles and zeroes (in the z-plane) can be mapped into the s-plane to determine the equivalent analog filter function, and vice-versa. The digital filter section of Eq. (9) corresponds to an analog biquadratic filter section with,

$$H(s) = \frac{s^2 + \alpha_0 w_0 s + w_0^2}{s^2 + \alpha_1 w_1 s + w_1^2} \qquad (11)$$

However, the periodic nature of the e^{sT} function causes the digital filter passband to repeat periodically. The effect is the same as aliasing. The analog filter response is mixed with the sampling frequency harmonics to generate the true digital filter response.

Designing Digital Filters

How does one go about designing a digital filter? One approach is to perform a least mean squared error optimization using a computer. The desired function is specified, and the computer adjusts the a_n and b_n values until the desired response is achieved.

A second approach is to design an equivalent analog filter and then convert that design to a digital filter. This approach has great merit, since analog filter design theory is well developed. However, the digital passband will be distorted if the analog equivalent filter has significant response to frequencies greater than $f_s/2$. The aliased passbands overlap at that point.

To circumvent this problem, the analog filter function can be modified to compensate for the aliasing effects. The analog transfer response is modified using several transforms to compensate for aliasing. Unfortunately, the nature of the s-plane to z-plane mapping is such that no transform can compensate for all aliasing effects without introducing other forms of distortion.

The **standard (or impulse-invariant) z-transform** represents a direct mapping to the z-plane. No frequency, amplitude, or phase distortion is introduced, but aliasing effects are not compensated. This transform should be used when the analog filter has negligible response to frequencies greater than $f_s/2$.

The **bilinear z-transform** preserves the filter amplitude response in the presence of aliasing. However, the bilinear transform introduces a distortion or **warping** of the frequency axis. As a result, only the filter cutoff frequency can be accurately transformed, using a **pre-warping** technique. Frequencies within the filter passband remain warped, introducing phase distortion in the digital filter response. The bilinear transform is used when the filter amplitude response is more critical than the phase response.

The **matched z-transform** preserves the filter phase response at the expense of amplitude response distortion. However, this amplitude distortion, unlike the aliasing distortion, can be corrected by placing additional zeroes in the transfer function. The matched transform is used when the filter phase response is critical, and either the amplitude response is not critical or the additional compensation zeroes can be accommodated.

Performing the transforms by hand is quite tedious. Fortunately, computer programs are widely available which handle the complete filter synthesis procedure, including z-transforms and pre-warping. One such program is FILSYN, written by Compact Engineering, Inc. and available on the National CSS timeshare computer service. Once the user has determined the desired filter response and z-transform method, FILSYN will perform all necessary synthesis to generate the digital filter coefficients.

Limit cycle noise

An effect of using digital numbers with a finite number of bits is the generation of quantization noise. When implementing digital filters, the quantization noise introduces oscillations that are analogous to ringing in analog filters. These oscillations are called **limit cycles.** The limit cycles generate a noise which peaks at frequencies corresponding to the filter pole frequencies. The noise power is roughly proportional to pole Q. Limit cycle noise for a second order filter section of Eq. (11) is given by (Ref. 1):

$$N_L(\mathrm{dB}) = 10\log_{10}\left(\frac{2}{12}2^{-2b}\,\frac{1+r^2}{1-r^2}\,\frac{1}{r^4+1-2r^2\cos 2w}\right) \qquad (12)$$

where b = number of digital number bits (excluding sign bit)

pole freq. = w_1

pole Q = $1/\alpha_1$

$$w = 2\pi\frac{\text{pole freq.}}{f_s} \qquad r = \exp\left(\frac{-w}{2 \bullet \text{pole Q}}\right)$$

The limit cycle noise must be calculated for each complex pole pair, and adjusted to reflect the response of subsequent filter stages to the limit cycle frequency. Computer programs like FILSYN can calculate limit cycle noise power, including all of these considerations.

Output signal reconstruction

Once manipulation of the digital sample numbers is complete, the resulting digital numbers must be converted back to a continuous analog signal. Referring back to Fig. 3, a digital-to-analog (D/A) converter transforms the digital numbers to a series of analog output pulses.

A sample-and-hold (S&H) circuit eliminates transients that are introduced during the D/A conversion process. The spectrum of the S&H output is modifed as follows:

$$\text{S\&H } F'(jw) = F(jw)\frac{t}{T}\,\frac{\sin(wt/2)}{wt/2} \qquad (12)$$

where t = hold time   T = sample period

An output smoothing filter completes the reconstruction by removing all components with frequencies greater than $f_S/2$. The smoothing filter is often optional, depending on the importance of removing the high-frequency output components.

The spectral effects of reconstruction are shown in Fig. 4.

DSP Components

Monolithic Memories, Inc manufactures a broad line of bipolar components ideally suited for DSP applications. In defining a suitable architecture for the spectrum analyzer, a familiarity with these components is essential.

Multipliers

Monolithic Memories offers three dedicated multiplier chips, with more to be announced in the near future. The chief characteristics of the Monolithic Memories' multipliers are given in Table I.

The 74S508 and 74S516 are sequential multipliers, while the 74S558 is a full parallel multiplier. The parallel unit is considerably faster, but the sequential units provide accumulation and division as well as multiplication functions. The 74S516 is particularly attractive, in that a complete 16-bit multiply/divide/accumulate function is provided in a single package.

The instruction set of the 74S516 is shown in Table II. The 74S516 multiply time of 1.25 μs may appear slow when compared to the multipliers in the single-chip DSP devices, but the added flexibility of the PAL approach more than compensates for the slightly reduced multiplier speed. In fact, this analyzer provides over three times the input bandwidth (determined by the input sampling rate) of comparable analyzers using the AMI S2811 or NEC μPD7720 and over six times the input bandwidth of an analyzer using the Intel 2920.

PALs

Monolithic Memories' programmable array logic (PAL) chips provide an excellent alternative to TTL when implementing random and control logic. The PAL is similar in concept to the PLA, in that logic is implemented by using an AND-OR matrix array. In the PAL, however, only the AND matrix is programmable. The OR matrix is fixed. The slight reduction in flexibility allows the addition of clocked registers, bidirectional input/output pins, exclusive-OR, and arithmetic functions on various PAL chips, all with faster propagation delay than PLAs. The PAL family is summarized in Table III.

The PALs are packaged in 20-pin "skinny dips." PC board area is minimized. The wide variety of PAL functions permits architecture optimization with a minimal total package count. PAL functional density is sufficient to compete favorably with bipolar bit-slice devices in providing a good function/speed/package count balance.

TABLE I
Monolithic Memories Multipliers

P/N	BITS	OPERATION	MPY TIME	PKG	No. PKGS. FOR 16x16	16x16 MPY TIME
74S508	8x8	SEQ.	750 ns	24-DIP	1	3500 ns
74S516	16x16	SEQ.	1250 ns	24-DIP	1	1250 ns
74S558	8x8	PAR	100 ns	40-DIP	10*	184 ns*

*includes partial product adder matrix using Monolithic Memories' SN74S381 and SN74S182 devices

TABLE II
MMI 74S516 INSTRUCTION SET

A0 – A2 SEQUENCE	OPERATION	CLOCK CYCLES
0	X1Y	9
1	–X1Y	9
2	X1Y + Kz, Kw	9
3	–X1Y + Kz, Kw	9
4	Kz, Kw/X1	21
5/6 0	XY	10
5/6 1	–XY	10
5/6 2	XY + Kz, Kw	10
5/6 3	–XY + Kz, Kw	10
5/6 4	Kw/X	22
5/6 5	Kz/X	22
5/6 6 0	XY + Z	11
5/6 6 1	–XY + Z	11
5/6 6 2	XY + Kz • 2^{-16}	11
5/6 6 3	–XY + Kz • 2^{-16}	11
5/6 6 4	Z, W/X	23
5/6 6 5	Z/X	23
5/6 6 6 0	XY + Z, W	12
5/6 6 6 1	–XY + Z, W	12
5/6 6 6 2	XY + W_{sign}	12
5/6 6 6 3	–XY + W_{sign}	12
5/6 6 6 4	W/X	24
5/6 6 6 5	W_{sign}/X	24
7	Read Z	1
7 7	Read Z, W	2
5 7	Round, then Read Z	2
5 7 7	Round, then Read Z, W	3

Table II Notes:

1. X, Y are input multiplier and multiplicand.
2. X1 is previous multiplier value.
3. 5/6 specifies input format. Use 5 for fractional arithmetic and 6 for integers.
4. Z, W is a double-precision number. Z is MSB. Represents addend on input and product (or accumulator sum) on output.
5. Kz, Kw represents previous accumulator contents. Kz is MSB.
6. W_{sign} is a single length signed number.
7. Each clock cycle = 167 ns for a 6-MHz clock.

FIFOs

The 67401 FIFO is a 64x4-bit device providing very fast (10 MHz) shift rates. The elastic storage provided by FIFOs permits sample rate reduction in the analyzer with minimal software overhead. Similar sample rate reductions cannot be accomplished using single-chip DSP devices, because of the very limited elastic storage provided.

The FIFOs maintain a uniform, jitter-free input and output sample rate for the analyzer. This is required by sampling theory. However, crunching of the digital numbers within the analyzer can be non-uniform, as long as sufficient FIFO elastic storage is provided. With FIFO buffers at the input and output, internal sample rates may be optimized, limited only by aliasing considerations.

PROMs

The 63xx series of bipolar programmable read-only memory (PROM) chips provide storage for filter coefficients and microprogram control. The 6309-1 PROM is a versatile 256x8-bit device, with a 70 ns worst-case access time, making it ideally suited for use in the analyzer.

Other Components

Monolithic Memories also second-sources a variety of Schottky TTL octal registers, octal bus buffers, and arithmetic chips. The spectrum analyzer can be efficiently implemented using the Monolithic Memories' components and only a handful of additional devices.

Implementing the Spectrum Analyzer

The architecture used for the spectrum analyzer is shown in Fig. 6. Input signals are digitized and buffered with FIFOs before interface with a common 16-bit data bus. The 16-bit arithmetic unit (AU) provides multiply and accumulate operations. A 16-bit wide RAM stores intermediate results. A 16-bit temporary register facilitates z^{-1} delays and data movement. Outputs are provided using a D/A converter and S&H circuits.

TABLE III
Monolithic Memories PAL-20 Family

P/N	INPUTS	OUTPUTS	BIDIRECTIONAL I/O	REGISTER BITS	FUNCTIONS
PAL10H8	10	8	—	—	AND-OR
PAL12H6	12	6	—	—	AND-OR
PAL14H4	14	4	—	—	AND-OR
PAL16H2	16	2	—	—	AND-OR
PAL16C1	16	1	—	—	AND-OR/NOR
PAL10L8	10	8	—	—	AND-NOR
PAL12L6	12	6	—	—	AND-NOR
PAL14L4	14	4	—	—	AND-NOR
PAL16L2	16	2	—	—	AND-NOR
PAL16L8	8	—	8*	—	AND-NOR
PAL16R8	8	8*	—	8	AND-OR
PAL16R6	8	6*	2*	6	AND-OR
PAL16R4	8	4*	4*	4	AND-OR
PAL16X4	8	4*	4*	4	AND-OR-XOR
PAL16A4	8	4*	4*	4	AND-CARRY-OR-XOR

*TRI-STATE OUTPUTS

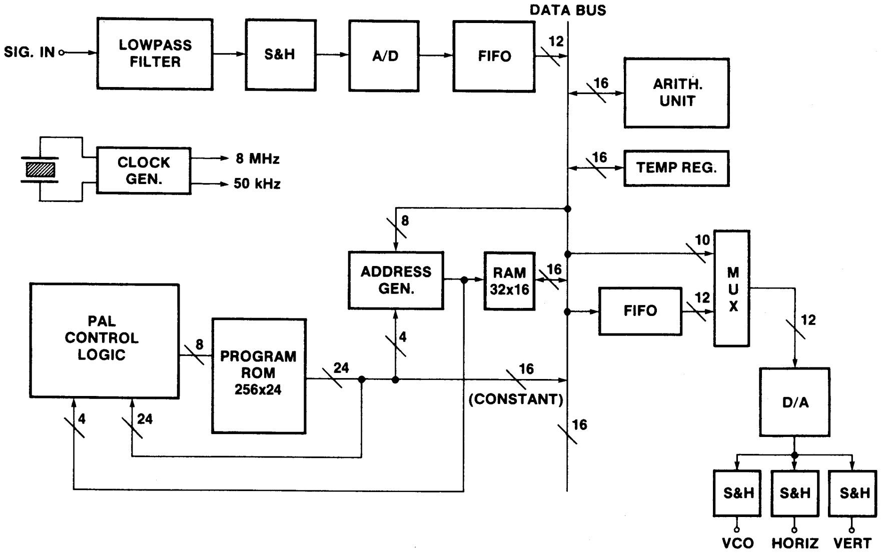


Figure 6. Analyzer Architecture

The VCO output is buffered using FIFOs to provide a uniform high-speed sample rate. The VCO output is 12-bits wide, providing a signal-to-quantization noise ratio of 91 dB, using Eqs. (5) and (7). The calculation assumes a 500-Hz bandwidth. A smoothing filter at the VCO output is not necessary. The filter test configuration of Fig. 2 allows the input aliasing filter to remove the effects of VCO high-frequency components, as long as the filter under test is a linear analog circuit.

The vertical and horizontal outputs are intended to interface an oscilloscope or X-Y plotter. The sampling of these outputs can be non-uniform, as long as the outputs track each other. The elastic storage at the input and VCO interfaces permits arbitrary non-uniform processing of the analyzer functions.

The 16-bit resolution of the internal data word provides 90 dB dynamic range according to Eq. (4), or 115 dB dynamic range according to Eqs. (5) and (7), assuming 500 Hz bandwidth and no limit cycle noise and aliasing.

Microprogram control was selected for the analyzer. PALs can efficiently implement sequential state machines. It is possible to encode all control information in the PALs, but only three packages would be saved (one PROM and two buffers). Distributing the control among several PALs would reduce flexibility and make corrections very difficult. The few extra packages required for microprogram control provide an extremely flexible architecture and greatly simplify the PAL functions.

With the theoretical background and architecture in mind, the spectrum analyzer functions can be defined in detail. The objective is to realize a circuit capable of high-resolution analysis of audio signals in the DC to 20-KHz range. Selectable bandwidth and linear/logarithmic output display are highly desirable. The detailed functions are shown in Fig. 7.

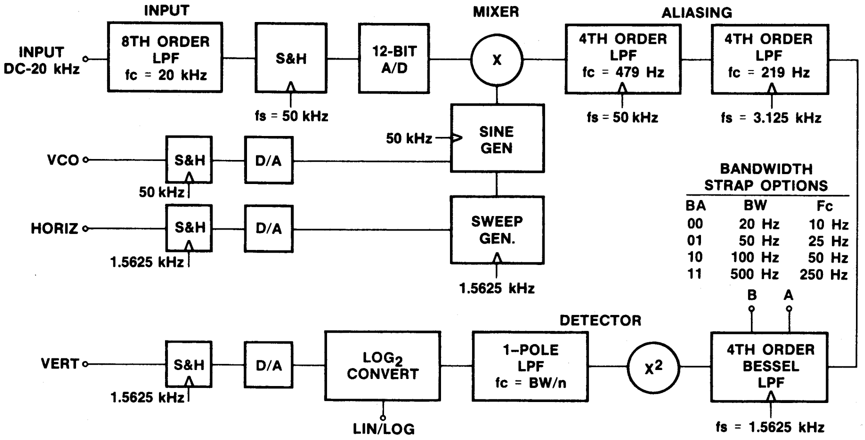


Figure 7. Detailed Functional Diagram

Input aliasing filter

An analog lowpass filter removes high-frequency components from the input signal. With a sample of 50 kHz and a maximum input frequency of 20 kHz, the lowest aliasing frequency is (50-20) = 30 kHz.

An eight-order Chebychev filter with 0.1 dB passband ripple will provide 44 dB of attenuation at 30 kHz, and 86 dB attenuation at 50 kHz. It is desirable to provide at least 60 dB overall dynamic range for high-performance analysis. To eliminate spurious responses above the –60 dB "floor," the input signal should have all components above 30 kHz suppressed by at least (60-44) = 16 dB. Most input signals will meet this requirement. If not, additional filtering must be provided.

S&H and A/D converter

The input S&H maintains a constant sampled signal level while the A/D conversion is in progress. No sin X/X correction is made for this S&H since the net effect of the S&H plus A/D action is an impulse sample at the start of the "hold" period.

The A/D conversion time should be less than 16 μs. The A/D converter output digital number should "saturate" when the input signal exceeds the maximum level. The digital numbers should be in inverted two's complement form. The S&H acquisition time should be less than 4 μs.

For a full 60 dB overall dynamic range, a 12-bit A/D is required. The 1980 **IC Master** lists several A/D converters meeting all of these requirements.

Mixer

The mixer multiplies the A/D output by the swept sinewave oscillator value. The multiplication produces sum and difference frequencies, according to Eq (1).

Two's complement fractional arithmetic is used throughout the analyzer. Multiplication cannot overflow, since all numbers are less than 1 in magnitude.

Swept sinusoidal oscillator (VCO)

A precision swept sinusoid from DC to 20 kHz must be generated to mix with the input signal. A technique particularly well suited to this application is solving the two equations:

$$\sin\ (x+y) = \sin x \cos y + \cos x \sin y \qquad (13)$$

$$\cos\ (x+y) = \cos x \cos y - \sin x \sin y \qquad (14)$$

These two trigonometric identities generate a new sin and cos value with y representing the phase shift per sample period. The technique is a "CORDIC" algorithm, based on coordinate rotation. Exact results are produced, but truncation and round-off errors due to the finite digital word length can cause a slow change or "drift" of carrier amplitude. Fortunately, the swept sinusoid is periodically reset in the spectrum analyzer, arresting this amplitude drift.

The VCO frequency is swept by varying the value of y. However, since Eqs. (13) and (14) require sin y and cos y, an identical CORDIC algorithm is used to obtain these values. To sweep the VCO, then, sinΔ and cosΔ are placed in RAM, selected by the bandwidth setting. These are two fixed numbers originally stored in PROM, and represent the frequency shift per sample time. Eqs. (13) and (14) are then applied to calculate sin y and cos y, which represent the desired phase shift per sample time. Eqs. (13) and (14) are executed again to generate the actual sinusoidal output.

The calculation of sin y and cos y can take place at a reduced sample rate to save processing time. Only the last execution of Eqs. (13) and (14) must be performed at the full 50-kHz sample rate.

A linear ramp is generated to provide horizontal drive for an oscilloscope or X-Y plotter. The ramp is incremented each time the sin y and cos y values are updated, tracking the VCO sweep. When the ramp value overflows, the analyzer sweep cycle is re-initialized.

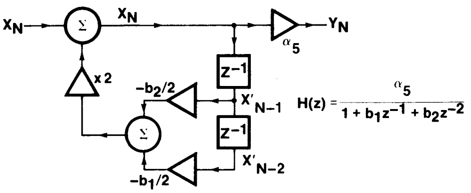


Figure 8. All-Pole Digital Filter Section

Digital filters

Fig. 8 shows the implementation of the all-pole digital filter sections. Because of the low pole Q values in all filters, the second order sections can be simply cascaded to implement high-order filters. Fig. 8 shows a technique for handling coefficients greater than 1 with fractional number representation.

Scaling must be performed to ensure maximum dynamic range. Filter sections with high-Q poles will show peaking of signals near the pole frequencies. The input to such sections must be scaled down to prevent overflow of the arithmetic. For a second-order all-pole section, this peaking factor is exactly the Q of the poles. Thus, when a given second-order section has a pole Q of 2, the input signal to that section must be multiplied by 0.5 to prevent overflow. When the Q is less than or equal to 1, no scaling is performed.

Saturation arithmetic is not provided in this architecture. Careful scaling eliminates the need for saturation arithmetic, since the A/D will saturate at a precisely known value.

The digital filters are synthesized and analyzed using the FILSYN program. FILSYN provides a limit cycle analysis of individual filter sections and of the overall filter response. FILSYN is a highly interactive program, and is quite straight-forward in its use. Use of such programs permits the design of digital filters with only a brief familiarity of the theoretical concepts involved.

Aliasing filters

Two 4th-order Chebyschev filters permit reduction of the sample rate following the mixer. Each filter provides 0.3 dB passband ripple and at least 68 dB attenuation of aliasing components. The slightly high passband ripple is acceptable, since subsequent filters will dominate the composite passband shape.

The first filter permits a sample rate reduction factor of 16. It is designed with a passband cutoff frequency of 479 Hz and a sample rate of 50 kHz. Eq. (12) predicts the limit cycle noise for this filter to be –58 dB.

The second filter permits a second sample rate reduction factor of 2. Its cutoff frequency is 219 Hz, with a sample rate of 3.125 kHz. Eq. (12) predicts the limit cycle noise for this filter to be –83 dB. This filter also provides an additional 14 dB attenuation of the limit cycle noise generated in the first aliasing filter, reducing the limit cycle noise from the first filter to –72 dB.

These two filters permit an overall f_S reduction factor of 32 before processing the Bessell filter, detector, and linear-to-$\log_2$ functions. This results in a very substantial throughput improvement. Net execution time is determined by the time to execute a given function multiplied by the sample rate for that function. Thus 32 instructions at the reduced rate will increase the net execution time by an amount equivalent to only 1 instruction at the full sample rate.

Fig. 9 shows the aliasing spectra of the sample rate reduction process.

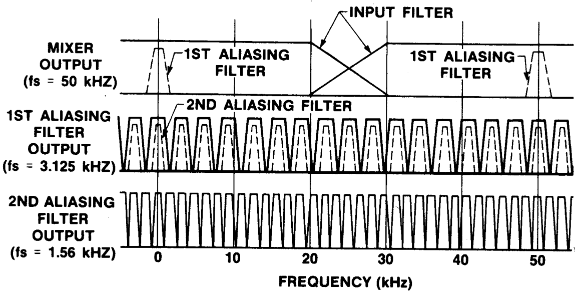

INPUT FILTER
MIXER OUTPUT (fs = 50 kHZ)
1ST ALIASING FILTER
1ST ALIASING FILTER
2ND ALIASING FILTER
1ST ALIASING FILTER OUTPUT (fs = 3.125 kHZ)
2ND ALIASING FILTER OUTPUT (fs = 1.56 kHZ)
0
10
20
30
40
50
FREQUENCY (kHz)

Figure 9. Aliasing Spectra for f_S Reduction

Lowpass filter

A 4th-order Bessel lowpass filter determines the overall bandwidth of the analyzer. The overall bandwidth is twice the bandwidth of this filter. Overall bandwidths of 20, 50, 100 or 500 Hz are provided by loading the proper set of filter coefficients into RAM when the bandwidth is selected.

The Bessel filter provides an optimal transient response for the analyzer. Good transient response is important, especially at narrow bandwidths, since the spectrial peaks are swept with respect to the filter passband. The net effect is similar to pulsing the filter input. Because the phase response is critical, the matched-z transform is used to convert the analog Bessel design to the z-domain.

The second aliasing filter provides 3 dB of attenuation at 250 Hz. When cascaded with the Bessel filter, which also provides 3 dB attenuation at 250 Hz in the 500 Hz bandwidth mode, the response at this bandwidth is modified. However, since the overall bandwidth is relatively broad, good transient response is still achieved. Cascading these two filters provides a "transitional" filter with a Bessel response at low attenuation and a Chebyschev response at high attenuation. At bandwidths less than 500 Hz, the combination produces an optimal tradeoff between transient response and resolution.

Analysis of Eq. (12) reveals that limit cycle noise increases exponentially as the pole frequency is reduced. Operating the lowpass filter at the lowest possible sampling frequency (1.5625 kHz) minimizes limit cycle noise, in addition to improving throughput. Limit cycle noise for the lowpass filter will be –95 dB at the 500 Hz bandwidth, and increases to –67 dB at the 20 Hz bandwidth.

Detector

A square-law detector provides a DC signal corresponding to the energy at the lowpass filter output. From trigonometry:

$$(A \cos wt)^2 = \frac{A^2}{2} (1 + \cos 2wt) \qquad (15)$$

The detector output contains the desired DC term and a single undesired term at frequency 2w. If the square law is ideal (easy in the digital domain), no additional terms are produced. The elimination of harmonics ensures the accuracy of the detector with $f_S/32$ = 1.5625 kHz. The highest component is always less than $f_S/64$ with a 250 Hz maximum lowpass filter cutoff frequency. However, 2w can be anywhere from DC to 500 Hz as the VCO sweeps past the spectral component.

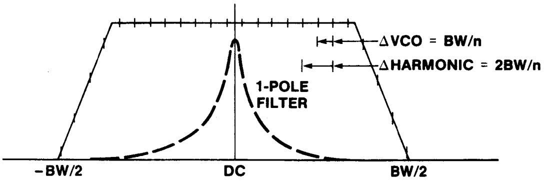


Figure 10. Detector Sweep Filtering

Fig. 10 illustrates a technique to render the effects of the 2w terms negligible. The analyzer passband is divided into n equal intervals. The VCO sweep rate is controlled so that the VCO sweeps BW/n per $f_S/32$ interval. The detector is followed by a single-pole lowpass filter with a 3 dB frequency of BW/n. As the VCO sweeps a component through the passband, the DC term is present in all n intervals, but the 2w term can affect only one interval. The worst-case DC error is 1/n for an ideal cutoff, and is multiplied by $(2^0 + 2^{-1} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-5.5} + 2^{-6} + 2^{-6.5} + 2^{-7} + 2^{-7.25} + \dots) = 1.85$ due to the finite 6 dB/octave rolloff of the single-pole filter. Further analysis reveals that:

$$n = \frac{1.85}{10^{e/10} - 1} \qquad (16)$$

where e represents the resulting error in dB. For e = 0.1 dB, n = 80.

In the filter test mode, the signal frequency and VCO frequency are the same, forcing w = 0. The detector has no error in this mode, but has a 3 dB gain due to the second DC term.

The detector output represents signal energy. Each bit in the detector output word thus represents only 3 dB, and 21 bits are required to reflect a 60 dB dynamic range. Double precision arithmetic is required for the detector ouput and the single-pole filter. The 74S516 multiplier will handle double precision calculations with a time penalty. Fortunately, the calculations to be performed are simple and the operations take place at the minimum sample rate, reducing the impact on throughput.

Linear to log conversion

The architecture of Fig. 6 is customized to provide an efficient algorithm for linear to logarithmic output conversion. The RAM address generator monitors the 8 MSBs of the data bus, and can provide a number indicating the MSB position of a positive number. This output is used to retrieve a lookup table value. This value is used to scale the data word to quickly left-justify the MSB. A second 4-point lookup table is then used to improve the accuracy of the resulting $\log_2$ conversion. A $\log_2$ conversion is adequate, since:

$$\log_{10} x = \frac{\log_2 x}{\log_2 10} \qquad (17)$$

Eq. (17) demonstrates that the output can be displayed in decibels by setting the oscilloscope or X-Y plotter Y-axis gain to the proper value.

Two lookup tables provide .027 dB accuracy for output values from 0 to –45 dB, and 3 dB accuracy from –45 to –84 dB. The logarithmic accuracy is limited by the 10-bit output word length to the D/A. This output can represent 0 to –84 dB in .082 dB increments. The accuracy of the 4-point lookup is therefore sufficient.

The logarithmic conversion procedure is as follows:

1) If the MSB of the data word is not among the 8 MSBs into the address generator, multiply the word by 2^7 = 128, and increment the output number by 7. Repeat until the data word is greater than 2^{-7}, but no more than three times. Set a flag if this step is executed more than once.

2) Look up the appropriate scale factor, from 2^0 to 2^6. Add $\log_2$ of the scale factor to the output word. The conversion is now accurate to 3 dB.

3) If the flag was not set during step 1, multiply the data word by the scale factor to left-justify the MSB position.

4) If the flag was not set during step 1, retrieve an intercept and slope value from the 8-word lookup table (four pairs available). Perform a linear interpolation using:

$$x' = a\,x + b$$

where a is the slope value (18)
b is the intercept value

The conversion is now accurate to .027 dB.

5) Scale the result to provide 84-dB output range with a 10-bit word.

6) Subtract 2^{-1} from the output to convert it to two's complement form for the D/A.

The calculations are double precision for steps 1, 2, and 3, and single precision thereafter. The conversion sequence can be bypassed using a strap option to provide a linear amplitude output from 0 to –30 dB.

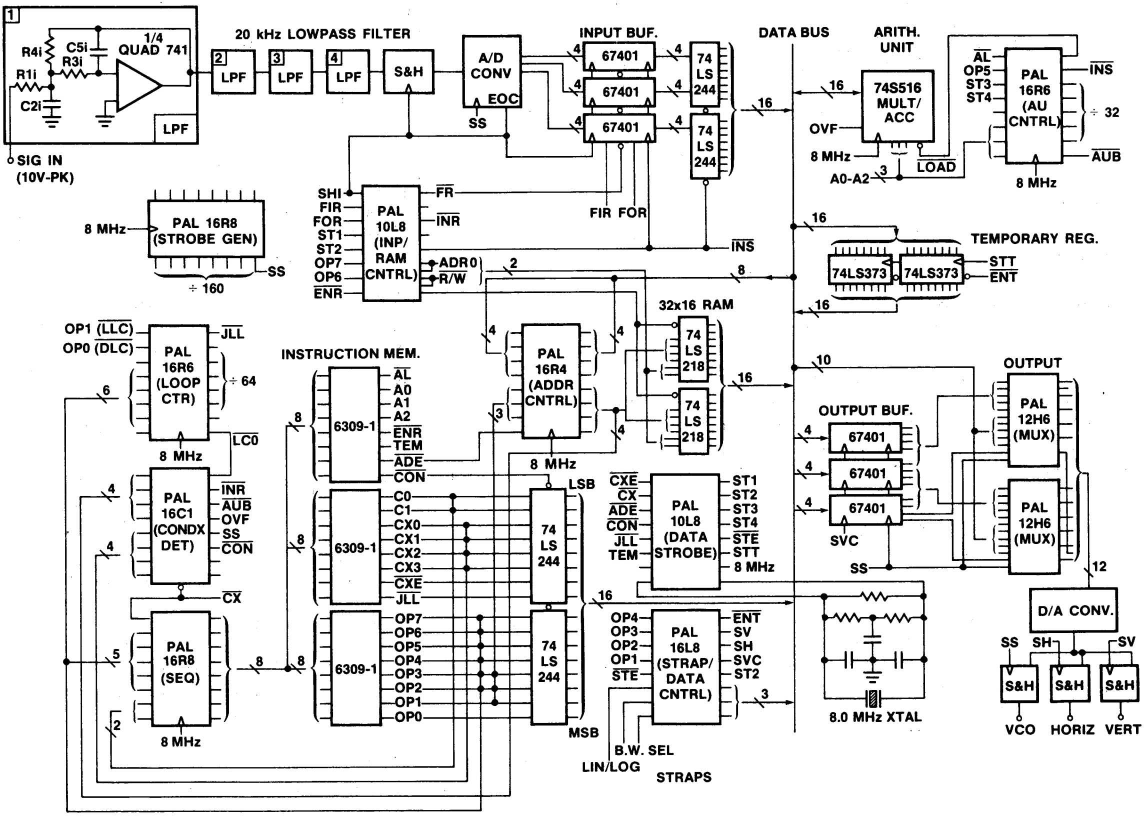


Figure 11. Simplified Schematic Hi-Fi Audio Spectrum Analyzer

Control logic

Fig. 11 shows a simplified schematic of the analyzer. All critical components are shown. Bypass capacitors and some component input connections are omitted for clarity.

The microprogram is stored in three 6309-1 PROMs. The microcode word formats are shown in Fig. 12. A wide, highly parallel instruction word ensures maximum flexibility and program efficiency.

Eight PAL devices interpret the instruction word and control the analyzer. Two additional PALs generate a 50-kHz strobe from the 8-MHz master clock, and implement the output D/A multiplexer. The control PALs function as follows:

Sequencer. A PAL16R8 implements an 8-bit instruction sequencer. A binary sequence would exhaust the available PAL OR-terms, so the sequencer is implemented using a 5-bit polynomial counter cascaded with a 3-bit binary counter. The sequencer performs the following operations:

C1	C0	$\overline{CX}$	Operation
0	0	X	Increment by 1 (execute next instruction)
0	1	0	Increment by 2 (skip next instruction)
1	0	0	Jump to address
1	1	0	No increment (repeat current instruction)

The $\overline{CX}$ input conditions the sequencer. Conditional branches or skip operations can be implemented. The sequencer will increment if the conditional requirement is not met.

A 5-bit jump destination can preset the polynomial counter. The binary counter stages are set to zero during a jump operation. Every eighth sequence address is, therefore, a possible jump destination. This compromise maintains adequate program flexibility, while permitting a single PAL to implement the sequencer.

Condition detector. A PAL16C1 monitors up to twelve status flags, and generates $\overline{CX}$. The microcode word

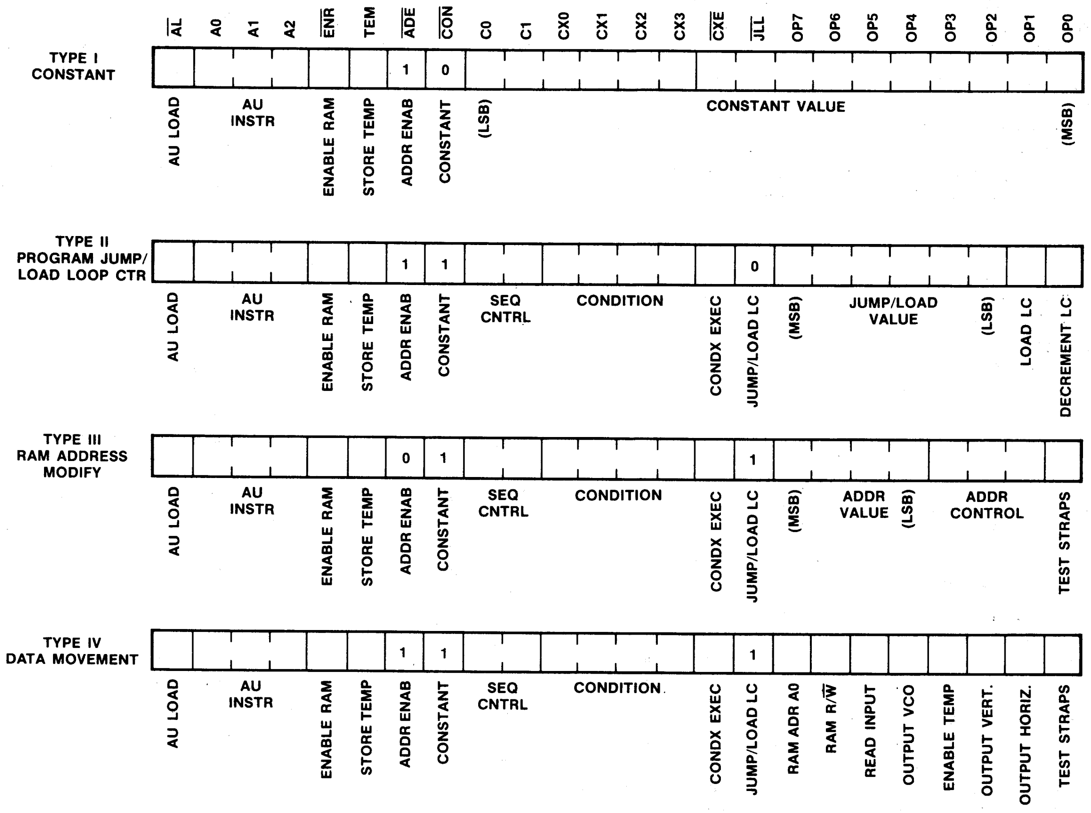


Figure 12. Microinstruction Word Formats

includes a 4-bit condition word, CX0 through CX3. $\overline{CX}$ will be zero under the following conditions:

CX3	CX2	CX1	CX0	Condition for $\overline{CX}$ = 0
0	0	0	0	Always (unconditional operation)
0	0	0	1	Sample strobe (SS) = 1
0	0	1	0	AU overflow (OVF) = 1
0	0	1	1	AU busy (AUB) = 1
0	1	0	0	Input sample ready (INR) = 0
0	1	0	1	Loop counter timeout ($\overline{LLC}$) = 0
0	1	1	0	$\overline{LLC}$ = 1
0	1	1	1	Address control (AC3-AC0) = 0
1	0	0	0	AC0 = 0
1	0	0	1	AC0 = 1
1	0	1	0	AC1 = 0
1	0	1	1	AC1 = 1
1	1	0	0	AC2 = 0
1	1	0	1	AC2 = 1
1	1	1	0	AC3 = 0
1	1	1	1	AC3 = 1

The assignment is made using the flexible PAL coding, and is optimized for the analyzer. The user can select a different set of conditions by re-programming the PAL.

When the microcode represents a constant (Type I microinstruction — see Fig. 12) the $\overline{CON}$ input forces $\overline{CX}$ = 1 to suppress conditional operations. $\overline{CX}$ is also used to suppress certain strobes in the analyzer, providing conditional arithmetic operations.

Loop counter. A PAL16R6 implements a 6-bit programmable down counter. This counter controls iteration loops and provides a timeout signal to the condition detector. The counter is preset via a Type II microinstruction, and can be decremented by other Type II microinstructions. The counter will halt when zero count is reached. Up to 64 iterations can be accommodated with minimal overhead.

Address control. A PAL16R4 provides indexed addressing for the 32x16 RAM, and analyzes the 8 MSBs of the data bus for conditional operations. If D15 represents the data bus sign bit, then OP1-OP3 will provide the following functions:

OP3	OP2	OP1	AC3-AC0 Output Function
0	0	0	Clear (0000)
0	0	1	Increment
0	1	0	Decrement
0	1	1	Preset to D15-D12 (Sign + 3 MSB)
1	0	0	Preset to D14-D11 (4 MSB)
1	0	1	Preset to D11-D8 (Address load)
1	1	0	MSB position
1	1	1	No change

The $\overline{ADE}$ input enables a change in the address word. The address word will not change if $\overline{ADE}$ = 1.

The MSB position function indicates the position of the MSB for positive numbers. AC3 represents sign bit D15. This output should be zero. AC2-AC1 represent the position of the first 1 following the sign bit. Code 0000 indicates that D15-D8 are all 0.

Input/RAM control. Miscellaneous FIFO input and RAM control is provided by a PAL10L8. The 67401 FIFO includes input ready (FIR) and output ready (FOR) signals, which are latched using the input/output shift clocks to generate two flags. The first flag ($\overline{FR}$) resets the FIFO when input ready (latched) goes low, indicating the FIFO capacity is exhausted. The latched output ready signal flag represents input sample ready ($\overline{INR}$). The INR flag is used as a sequencer condition to synchronize wait loops. Use of the FR and INR flags maintains proper fill of the FIFO.

The RAM address LSB (A0) and read-write line (R/$\overline{W}$) are decoded and latched. These signals are provided directly by Type IV microinstructions.

Notice that a clocked register function requires two PAL combinatorial outputs per bit, while a transparent latch function requires only one PAL output per bit.

Arithmetic unit control. The variety of functions listed in Table III indicate the utility of the 74S516 arithmetic unit (AU). A PAL16R6 complements the 74S516 to provide simplified control of the AU.

The PAL gates and AU load signal provide conditional arithmetic operations. Gating the 67516 load input will suppress the start of a new arithmetic operation. When $\overline{CXE}$ is high, the operation is performed unconditionally. When $\overline{CXE}$ is low, the operation is performed only if $\overline{CX}$ is low. Combining conditional jumps and conditional AU operations provides a high degree of program flexibility.

The PAL monitors the AU instructions and generates a busy signal ($\overline{AUB}$). A counter in the PAL keeps track of variable-length operations to provide the correct output for any instruction sequence. The $\overline{AUB}$ signal conditions the sequencer to synchronize the microprogram to the AU operation. Microprogramming is simplified as a result.

The PAL also gates the input FIFO shift out clock ($\overline{INS}$) to eliminate transients while providing a full 125 ns pulse for proper FIFO operation.

Data strobe generator. A PAL10L8 provides a number of transient-free, gated strobes. These strobes provide control of the analyzer data flow. The PAL gating interprets the microinstruction to determine the proper microinstruction type, and generates the strobes accordingly.

The PAL also generates an 8-mHz buffered clock, as shown in Fig. 11. The crystal oscillator circuit provides independent AC and DC feedback, permitting reliable operation with the PAL.

Strap/output sample control. A PAL16L8 generates additional control strobes for the output sample-and-hold circuits.

The PAL also provides a tri-state buffer function, connecting control straps to the data bus for certain conditional jump operations. Two straps select the desired analyzer bandwidth/sweep rate, and the third strap selects linear or logarithmic output.

Programming the PALs. The PALs provide a wide variety of control functions that are optimized by user programming. Several methods are available to allow easy PAL programming.

Programming is accomplished by blowing fuses in the PAL AND-array. Each AND-term contains the true and complement of all inputs. When no fuses are blown, the AND-term is disabled, since $A \bullet \overline{A} = 0$. AND-terms are activated by blowing fuses to disconnect unwanted inputs. The AND-terms are combined using OR, NOR, or exclusive-OR gates (depending on the PAL type) to generate the PAL outputs. The result is a sum-of-products Boolean expression representing the desired logic function.

The sum-of-products Boolean expressions are defined using standard digital logic design procedures. Combinatorial functions are plotted on Karnaugh maps and minimized. State tables and flowcharts are prepared for sequential state machines.

Once the state tables are minimized, the state variable transition functions are plotted on Karnaugh maps and minimized. The details of this procedure are described in the literature (Refs. 3, 4, 5).

Once the Boolean expressions have been derived, PALs may be obtained using several procedures:

1) Provide Monolithic Memories with a coded sheet indicating the desired fuse pattern. Blank coding sheets are available from Monolithic Memories.
2) Prepare an ASCII paper tape using either a "BPNF" or a hexadecimal format to represent the pattern, and mail or TWX the pattern to Monolithic Memories.
3) Obtain unprogrammed PALs and blow the fuses using a bipolar PROM programmer with an appropriate personality module. One such unit, designed specifically for PAL programming, is the Model SD-20 from Structured Design, Inc., 330 N. Mathilda Avenue, MS112, Sunnyvale, CA 94086.

Monolithic Memories offers PALASM, an assembler to generate PAL fuse patterns or paper tape files from Boolean expressions. This program is available from Monolithic Memories via the National CSS timeshare computer network. PALASM is an interactive, easily used tool for rapid pattern generation. The SD-20 programmer software incorporates the essential features of PALASM in its operation.

The programming procedure is thoroughly described in the **PAL Programmable Array Logic Handbook,** available from Monolithic Memories. A FORTRAN source listing of PALASM is included in the handbook.

Signal output

The VCO output must be sampled at precise intervals to avoid phase modulation effects. Three 67401 FIFOs buffer the VCO samples, which are generated during the 50-kHz input processing. A 12-bit D/A converter provides better than 91 dB signal-to-distortion ratio. The S&H circuit provides VCO outputs at precise 50-kHz intervals, and removes spikes that are generated in the D/A converter. All necessary control signals are generated by the strap/output data control PAL.

The horizontal and vertical outputs normally drive an X-Y plotter or oscilloscope. There is no need to buffer these signals as long as the two outputs track each other. The D/A used for the VCO output is shared by adding two PAL12H6 chips programmed as multiplexers. Use of PALs requires fewer packages than a TTL multiplexer. Additional S&H circuits decode the multiplexed D/A output to separate the output signals.

Microprogram

The architecture can implement a variety of DSP functions. A microprogram, stored in 6309-1 PROMs, customizes the architecture to perform the spectrum analyzer tasks. The microinstruction formats were summarized in Fig. 12. The algorithms to be implemented were discussed in the previous section. The step-by-step implementation of these algorithms is converted to a sequence of microinstructions to form the microprogram. The procedure is analagous to programming a microprocessor.

Operation of the microprogram is better understood by considering the allocation of the 74LS218 RAM locations, shown in Fig. 13. The microprogram consists of two parts. High-speed

(A4-A1)

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	(A0)
▨ –	▨ –	SIN X	SIN Y	SIN∠	b1	b1	–	x_{N-1}	x_{N-2}	x_{N-1}	x_{N-2}	x_{N-1}	x_{N-2}	x_{N-1}	x_{N-2}	0
▨ –	▨ –	COS X	COS Y	COS∠	b2	b2	–	x_{N-1}	x_{N-2}	x_{N-1}	x_{N-2}	X	X	X	VCO RAMP	1

CARRIER GENERATOR LOWPASS FILTER ALIASING FILTERS LOWPASS FILTER

(a) INPUT

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	(A0)
▨ TEMP LSB	▨ FLAG	X	X	SCALE A	SCALE B	SCALE C	SCALE D	X	X	X	X	X	X	X	X	0
▨ TEMP MSB	▨ OUTPUT	X	X	LOG A	LOG B	LOG C	LOG D	X	X	X	X	x_{N-1} LSB	x_{N-1} MSB	b1	X	1

UTILITY LIN-LOG SCALING DETECTOR

(b) OUTPUT-1

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	(A0)
▨ TEMP LSB	▨ FLAG	X	X	a_1	a_2	a_3	a_4	X	X	X	X	X	X	X	X	0
▨ TEMP MSB	▨ OUTPUT	X	X	b_1	b_2	b_3	b_4	X	X	X	X	X	X	X	X	1

UTILITY LIN-LOG INTERPOLATION

KEY: ▨ SCRATCH X NOT AVAILABLE
– NOT USED □ TABLE

(c) OUTPUT-2

Figure 13. RAM Allocation

input processing provides the carrier generation, mixing, aliasing filter and lowpass filter functions. Fig. 13(a) shows the RAM allocation during input processing. The input segment includes an efficient iteration loop, using the PAL loop counter, to process the 50-kHz functions. The carrier frequency shift and lowpass filter functions are processed at the $f_S/32$ reduced sample rate for maximum throughput efficiency.

The values of sin Δ, cos Δ, and the Bessell filter coefficients depend on the analyzer bandwidth strap selection. These values are stored in a "table" area in Fig. 13, and can be easily changed. The fixed aliasing filter coefficients are stored as constants in the microprogram itself.

Once the input processing is complete, coefficients located in the table area can be changed. This area is re-used by the output program segment to hold the scale factors for the linear-to-log conversion routine. The detector functions are processed, and the logarithmic conversion is started with the RAM allocation of Fig. 13(b). The table area is then reloaded with the interpolation coefficients (Fig. 13(c)) to complete the logarithmic conversion. Shaded areas in Fig. 13 provide temporary data and flag storage for the routines.

The microprogram samples the strap settings and loads the table area with the appropriate coefficients for input processing. The detector filter coefficient (b_1) is also determined and loaded. The input processing is then repeated. This sequence repeats indefinitely. The coefficient loading technique makes efficient use of RAM capacity while eliminating elaborate jump sequences. All coefficient table updates are processed at the minimum sample rate for best efficiency.

The PAL controllers simplify the microprogram. A PAL provides hardware iteration loops. The AU controller eliminates wasteful "NO-OP" instructions otherwise required to allow completion of AU operations. The input control PAL simplifies handshaking with the input logic. With the benefit of the PAL controllers, the analyzer microprogram easily fits into the 256-instruction capacity of the PROMs.

Performance

Table IV summarizes the performance of the spectrum analyzer. The performance figures indicate that the 67516-based analyzer is quite useful for high-fidelity audio work. Analyzer performance using other Monolithic Memories multipliers is given in the table. Performance of a spectrum analyzer using the Intel 2920 (Ref. 6) is also shown as an example of the capability of a single-chip processor.

The bipolar approach provides substantial performance improvement, at the expense of increased package count. The 74S516-based analyzer provides over six times the input bandwidth of the 2920-based analyzer, even though the 2920 multiply time is only 20 percent slower than the 74S516. The performance difference is due to restrictions in the 2920 architecture that preclude generation of precision sinusoids at high speed and limit the ability to use sample rate reduction. These restrictions require use of a mixing scheme that must operate with bandwidths much less than the Nyquist requirements, and also precludes sample rate reduction. The performance advantages of a flexible architecture are clearly shown in this case.

Other single-chip DSPs present different problems. Unpublished analyzer benchmarks using the AMI S2811 reveal that while sampling rates in excess of 40 kHz are possible with this part, the internal 12-bit word length of the S2811 limits dynamic range to about 46 dB due to limit cycle noise. The 2920's 25-bit word length avoids this difficulty, but the 9-bit 2920 A/D resolution again limits dynamic range to 48 dB. To achieve both the high speed and performance required for high-fidelity analysis, therefore, the flexibility of the bipolar approach is required.

The bipolar flexibility is apparent when considering options available with the analyzer architecture. Use of faster bipolar multipliers can provide input bandwidths over 125 kHz (see Table IV), at the expense of a higher package count. Pipeline techniques can be added to achieve even greater bandwidth. Dynamic range can be improved by using a wider data word, again at the cost of additional packages. Designers can trade off performance versus complexity to get the job done.

Analyzer sweep rate and accuracy can also be traded off. The sweep rate for the 74S516 analyzer is (BW/n) × 1.56 kHz/sec, where BW is the bandwidth and n is given by Eq. (16). For example, the sweep rate with a 500-Hz bandwidth can be increased to 97.6 kHz/sec by relaxing the accuracy requirement to 1 dB. The resulting 4.88-Hz refresh rate for a full 20-kHz sweep is readily viewed on an oscilloscope.

The bipolar analyzer combines the high performance of MSI components with the flexibility and programmability of single-chip processors. The ability to customize the architecture provides yet another degree of flexibility. PAL controllers can be used to implement the analyzer in less board space than would be required with bit-slice components. The Monolithic Memories' components provide an efficient, cost-effective alternative in many digital signal processing applications.

TABLE IV. ANALYZER PERFORMANCE SUMMARY

	ANALYZER IMPLEMENTATION			
	MMI 74S508	MMI 74S516	MMI 74S558	INTEL 2920
fs (maximum)	21 kHz	56 kHz	356 kHz	16 kHz
Input bandwidth	7.5 kHz	20 kHz	125 kHz	3 kHz
Resolution	20-500 Hz	20-500 Hz	20-500 Hz	100 Hz
Accuracy	0.1 dB	0.1 dB	0.1 dB	1 dB
Dynamic range	60 dB	60 dB	60 dB	48 dB
Sweep rate (kHz/sec)	.4 – 9.8	.4 – 9.8	.4 – 9.8	6
Packages	34	34	50	2

References

1. A. Oppenheim, R. Schafer, **Digital Signal Processing,** 1975, Prentice-Hall, Englewood Cliffs, NJ.
2. J. Birkner, **PAL Programmable Array Logic Handbook,** Monolithic Memories, Inc.,
3. S. Unger, **Asynchronous Sequential Switching Circuits,** 1969, Wiley-Interscience, New York, NY.
4. F. Hennie, **Finite-State Models for Logical Machines,** 1968, John Wiley & Sons, New York, NY.
5. C. Clare, **Designing Logic Systems Using State Machines,** 1973, McGraw-Hill, New York, NY.
6. R. Holm, J. Rittenhouse, "Implementation of a Scanning Spectrum Analyzer Using the 2920 Signal Processor," Intel Corporation Application Note AP-92.

Notes

High-Quality Musical Sound Generator

The recent production of fast multiplier ICs and large PROMs as well as the low cost of MSI TTL has made possible the development of a digital symphony or large group of new musical sounds. This tutorial article describes a few basic acoustic parameters of musical sound. Then a digital architecture is developed for the synthesis of a modest sized orchestra. A minimal amount of musical knowledge is required to read this article.

Monolithic Memories

High-Quality Musical Sound Generator

A musician may use this timbre generator for the live performance or composition of music like that from a choir or modest sized orchestra.

Introduction

The recent production of fast multiplier IC s and large PROM s as well as the low cost of MSI TTL has made possible the development of a digital symphony or large group of new musical sounds. This tutorial article describes a few basic acoustic parameters of musical sound. Then a digital architecture is developed for the synthesis of a modest sized orchestra. A minimal amount of musical knowledge is required to read this article.

Overview

Music is crudely perceived as sequences of time-varying pitches of varying loudness. However, the timbre or tone color is an imimportant (yet poorly understood) aspect of music. "The character of the temporal evolution of the spectral components is of critical importance in the determination of timbre" [Risset and Mathews 1969]. With additive (Fourier-like) synthesis, the amplitude of each spectral component may be independently varied; and the frequency of each of these sinusoidal components may be independently varied. Dynamic spectra may be more efficiently computed with FM synthesis.

An oscillator is described in this article which is capable of generating sound with these synthesis techniques. This digital oscillator will directly generate 222 low disortion sine waves in realtime with a 32 kHz sample rate (or 189 sinusoidal components at a 40kHz sample rate). Using FM synthesis, thousands of sinusoidal components may be generated with this oscillator. Envelope generators are included which are capable of producing a maximum of 256 different amplitude envelopes (of any shape) and 256 different frequency envelopes. A pseudo random number generator is included for adding a small human like variation to constants. Random numbers may be sent through the oscillator to generate filtered noise of variable amplitude (for breath noise in wind instruments, bow noise for violins and cellos, or for percussive sounds). This module may be played (with a keyboard, joystick, pressure sensitive pads, etc.) by a musician in live performance or used with a disk drive for studio "multitracking" of control parameters [Mathews and Moore 1970].

Timbre

Timbre is that aspect of musical sound that causes a tone played on a violin, for example, to sound differently from a tone played on a flute when the same note at the same loudness level is being played on both musical instruments. When someone states that a piano sounds differently from a French horn, they mean that the timbre of the piano is different from the timbre of the French horn. Whether one desires to play with new timbres or the timbres of traditional music instruments, it is instructive to analyze the sound from acoustic musical instruments. Once the amount of detail of the variation of the parameters of acoustic musical sounds has been determined, these parameters may be played with or varied with enough detail to generate interesting non-traditional sounds.

The different musical instruments' timbres have often been (somewhat naively) attributed to the sound waveshapes of the different acoustic musical instruments. A clarinet tone will produce a different waveshape than a waveshape from a cello tone. But the waveshape changes throughout the duration of each note of one acoustic instrument. Adjacent tones in the scale of one musical instrument (such as a bassoon) will frequently produce considerably different waveforms [Backus 1969]. The ability to continuously vary the tone color (continuously variable waveshape) enables a good musician to add expression to each note. A constant square wave, sawtooth wave, triangle wave, or any unchanging waveshape sounds artificial, synthetic, or unnatural. A non-varying waveshape results from a mix of constant relative amplitudes and phases of harmonics. Harmonics are sinusoidal components (sine waves or cosine waves) whose frequencies are related by integers (1, 2, 3, 4, ...).* All audible waveshapes are composed of a sum of sinusoidal components. The sine wave (or cosine wave) is the only waveshape composed of no other sinusoidal component than itself. In acoustic musical instruments the amplitudes of the harmonics change independently of each other as a function of time as shown in Figure 1. An analysis of a typical clarinet tone is shown in Figure 1a [Moorer, Grey, & Strawn 1977] and an analysis of a violin tone is displayed in Figure 1b [Moorer, Grey, & Snell 1977]. These tones were analyzed using a heterodyne filter [Moorer 1973, 1975]. For an extensive description of timbre using these analyses see [Grey 1975]. The overall amplitude of a constant waveshape may be changed, but the amplitudes of its harmonics may not be independently varied.

Most acoustic musical instruments produce slightly inharmonic partials. Partials are sinusoidal components with frequencies of *any* values, while the frequencies of harmonics are limited to

**If the fundamental frequency (which is usually perceived as the pitch) of the note is 100 Hz, then the first harmonic will be 1 × 100 Hz = 100 Hz; the second harmonic will be 2 × 100 Hz = 200 Hz; the third harmonic will be 3 × 300 Hz = 300 Hz, etc.*

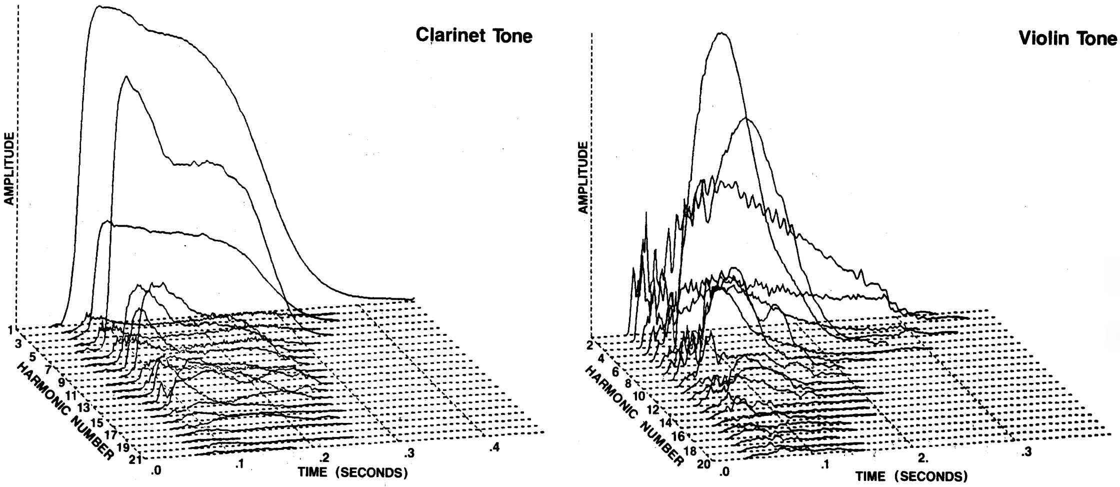


Figure 1. The Analysis of the Tone From a Single Note Played on a Clarinet, Then on a Violin. Each harmonic has its own characteristic amplitude envelope.

integer multiples of the fundamental frequency. For example, the frequencies of the first five partials of a tone played from the A string of an acoustic guitar are: 300 Hz, 595.4 Hz (= 1.985 × 300 Hz), 897.0 Hz (= 2.99 × 300 Hz), 1198.1 Hz (= 3.994 × 300 Hz), and 1500.0 Hz (= 5 × 300 Hz). The first five partials of a bell tone are: 278 Hz, 467 Hz, 620 Hz, 786 Hz, and 1046 Hz [Benade 1976]. The partials from the guitar tone are only slightly inharmonic, while those from the bell tone are strongly inharmonic. A constant waveshape has only harmonically (F, 2 × F, 3 × F, 4 × F,...) related partials.

The sound from most acoustic musical instruments is not perfectly periodic as may be seen from Figure 2. The analysis of the first partial of the tone from a single note played on a violin without vibrato is shown in Figure 2a [Moorer, Grey, & Snell 1977]. The amplitude and the frequency are displayed as functions of time. Figure 2b displays the analysis of the first partial of the tone from a single note played on an oboe [Moorer, Grey, & Strawn 1977]. In order to account for the quasiperiodic, inharmonic nature of partials, the frequency of each partial must be independently variable (frequency enveloped).

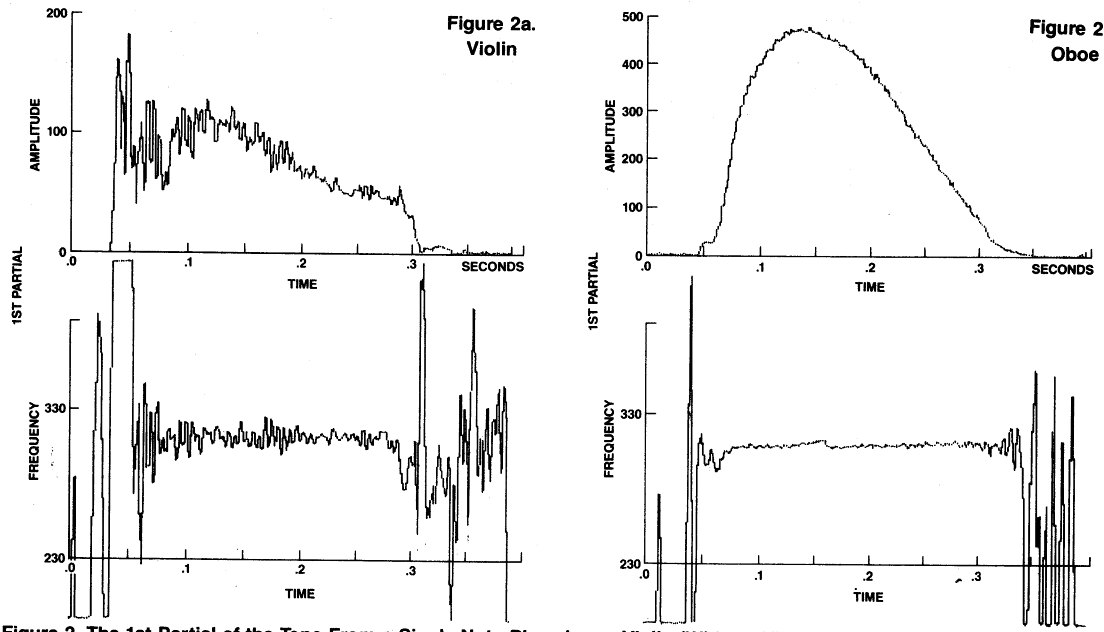


Figure 2a.
Violin

Figure 2b.
Oboe

Figure 2. The 1st Partial of the Tone From a Single Note Played on a Violin (Without Vibrato), Then on an Oboe. The tone is heard as a constant pitch, even though the frequency varies somewhat.

Simpler Control of Timbre

For some composers, additive synthesis (independent control of the frequency and amplitude of each partial) requires the control of too many parameters per voice (harp, viola, the human voice, or the sound of a new musical instrument). A simpler set of timbre controls is often desired, without sacrificing the rich tone color which results from a dynamically changing spectrum. It is also desirable to reduce the number of oscillators needed per voice, so that more voices may be generated without adding extra hardware. Many composers look to digital sound generators for the making of music like that of a large choir and orchestra, requiring an enormous number of partials. For these reasons, several more efficient synthesis techniques have been conceived [Moorer 1977, Le Brun 1977]. Presently, the most popular digital synthesis technique is the frequency modulation (FM) algorithm [Chowning 1973].

FM involves using one oscillator to modulate the frequency of another oscillator. If the output of a sine wave oscillator (the modulator) with frequency "F" is added to the frequency control (which is also set at "F") of a carrier sine wave oscillator, harmonics will be produced. The number of harmonics is directly proportional to the amplitude of the modulating sine wave. With a small amplitude modulating sine wave, only a few harmonics will be produced. The amplitude of the modulating oscillator is called the index of modulation and is effectively a bandwidth control. The ratio of the carrier frequency (c) to the modulating frequency (m) determines which partials are generated. A c/m ratio of irrational numbers c/m = $1/\sqrt{2}$, 1.01/1, $\pi/\sqrt{3}$, etc.) will result in inharmonic spectra. For more details read [Chowning 1973].

Thus a large number of partials may be generated with only a few controlling parameters and a minimum of two oscillators. Very natural timbre may be generated by using several oscillators (of different frequencies and small modulation indexes) to modulate one or two carriers [Schottstaedt 1977]. Whether additive synthesis, FM synthesis, or one of the more recent summation formulas is used, many sinusoidal oscillators are needed to generate a complete richly varying voice. Each oscillator requires an independent frequency control and an independent amplitude control. Recent investigation has indicated that a thousand (or more) partials are needed in the generation of full symphonic music.

Digital Oscillator Basics

A sine wave may be represented digitally as a series of pulses or steps as shown in Figure 3. If the steps or changes in pulse amplitudes are made infinitely small, a smooth analog waveshape will result. The steps should be made small enough so that the ear is incapable of hearing the difference between a smooth analog waveshape and the digital waveshape. Thus there will be no audible distortion or noise. Each of these steps is a pulse whose amplitude may be represented by a binary number in a memory. If one cycle of a digitized waveform is stored in memory (called the waveform memory), the numerical values of the pulse amplitudes may be read out by repeatedly incrementing the memory address (which corresponds to the phase angle of the sine). When the end of the waveform cycle is reached, the memory address will jump back to the beginning of the waveform memory. In other words, every time the phase angle is incremented to a value greater than or equal to 360°, 360° will be subtracted from the phase angle. Then the phase angle will continue to be incremented as before. If this waveform memory output is fed to a digital to analog converter (degliched)* which is followed by a smoothing filter, amplifier, and loudspeaker, a continuous sine wave will be heard. The digital to analog converter (DAC) changes the numbers into pulses whose amplitudes are controlled by the numbers readout of the waveform memory. The smoothing filter (low pass filter) rounds off the pulses so that a continuous waveform results as shown in Figure 3.

**To avoid a noisy output the digital to analog converter (DAC) is first followed by a sample and hold device or a track or ground switch. This will eliminate the output spikes which occur when the input word to the DAC is changed.*

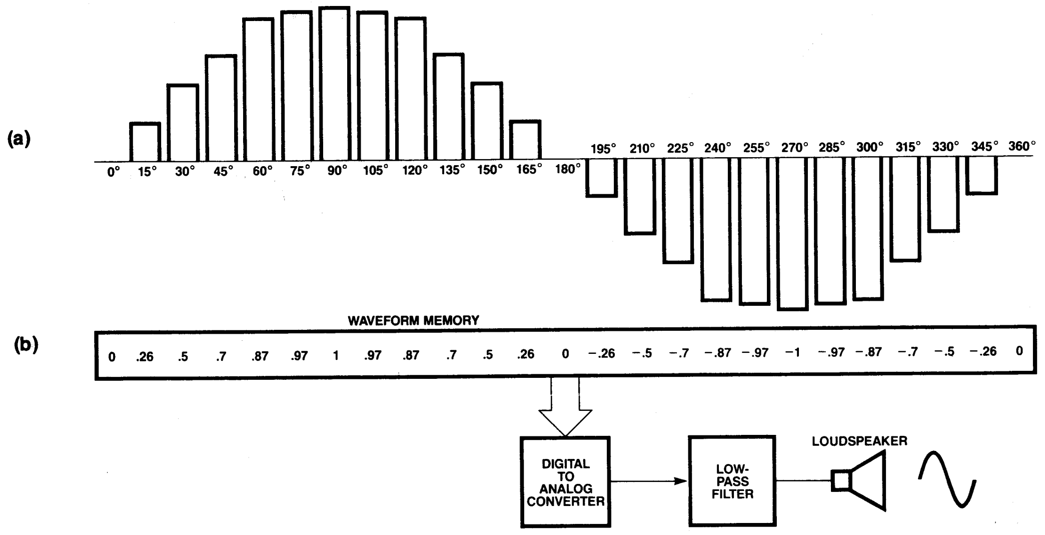


Figure 3. (a) Digitized Sine Wave. (b) Simple Tone Generation.

The frequency of the output waveform will be equal to the rate at which these pulses are generated (called the sample rate) divided by the number of pulses in one waveform cycle or period [Mathews, Miller, Moore, Pierce, and Risset 1969].

$$\text{frequency} = \frac{\text{sample rate}}{\text{number of samples per waveform cycle}}$$

The sample rate is crystal controlled for ultrastable tuning and is constant. If every number in the waveform memory is generated one after another at this constant sample rate, the frequency of the sine wave will be constant. This would result from incrementing the address of the waveshape memory by 1. If a higher frequency is desired, the address may be incremented by a larger number. If a lower frequency is desired, the address may be incremented by a smaller number. The phase angle is truncated to form the address for the waveform memory as illustrated in Figure 4. The phase angle increment register holds the value of the increment to the phase angle. This phase angle increment will be added to the phase angle once for each sample of the sine wave. When the address is near the end of the waveform memory (360°), an increment may be added which will overflow the phase angle register, thus placing the address back at the beginning of the waveform cycle. This is equivalent to subtracting 360° from the phase angle every time it is incremented to a value greater than or equal to 360°.

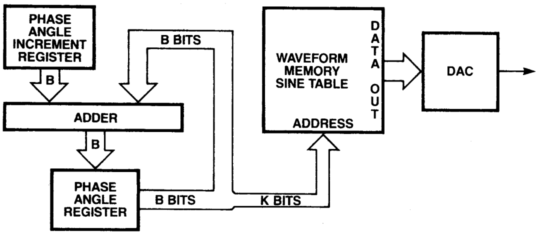


Figure 4. Variable Frequency Sine Wave Generation

Sampling Theory

At first glance, one might think that this technique would result in a large amount of distortion if the phase angle increment were as large as 180°. This would result in a frequency of ½ of the sample rate. With this phase angle increment value, only 2 pulses would be generated per waveform cycle. It can be shown mathematically [Carlson 1968] that if a signal contains only frequency components whose absolute values are less than some maximum frequency F_{max}, the signal may be completely described by the instantaneous sample values uniformly spaced in time with period $T_S \leq 1/2F_{max}$. If S is the sample rate, the sample period is $T_S = 1/S$. Alternatively, the Nyquist Sampling Theorem states that if a signal has been sampled at the Nyquist rate of $2F_{max}$ or greater ($S \geq 2F_{max}$), and the sample values are periodic weighted impulses, the signal can be exactly reconstructed from its samples by an ideal low pass filter of bandwidth B where $F_{max} \leq B \leq (S\text{-}F_{max})$.

Let's look at the problems that arise when we apply this theorem to generating sound. Any practical (non ideal) low pass filter will require a transition band to go from unity gain to 0 gain as shown in Figure 5. To avoid distortion; the maximum output frequency should be below this transition band. The maximum sinusoidal component frequency is usually limited to 40% of the sampling rate ($F_{max} \leq 0.4S$).

It might be noticed that realizable samples from the DAC will not be impulses; they will have finite pulse widths. If the pulse widths are as wide as the sampling period, the frequency response will be attenuated very little if at all for very low frequencies and attenuated up to 4 dB down, at frequencies which approach ½ of the sample rate. The reconstruction filter (low pass filter) may be designed to compensate for this. A different reconstruction filter should be used for each sample rate anticipated.

The theorem states that the samples from the DAC should be uniformly spaced, so the sample period jitter should be minimized. If the sample period jitter error is to be 82.6 dB below the *rms* level of the sine wave, the sample period jitter should be less than $(1.18 \times 10^{-5})/F$ where F is the output sine wave frequency [Talambiras 1977]. So if F is around 16 kHz, the sample period jitter should be less than 0.73 nanoseconds.

The ultimate test of a sound synthesis technique is listening with one's ear. The Soundstream Digital Recorder [Warnock, 1976] designed by Thomas Stockham, Jr. has provided one of the most convincing demonstrations of the audio application of the Nyquist Sampling Theorem. It has played back extremely clean, undistorted music.

Frequency Control

The frequency "F" of the sine wave is determined from the following equation:

$$F = \frac{I}{2^b} S$$

Where I is the phase angle increment $-180° \leq I < 180°$.
b is the number of bits in the phase angle register.
S is the sample rate.

The frequency is independent of the size of the sine wave memory. The number of bits (b) in the phase angle register and the sample rate (S) are constant as determined by the hardware. So the frequency may be varied by controlling only one parameter, the phase angle increment (I). The frequency resolution is determined by the number of bits in the phase angle increment register and the phase angle register. For smooth sounding portamento* and "pitch blending" 20 bits are needed in these registers [Snell 1977]. (This assumes no shifting or floating point techniques are used.) The use of 20 bit registers will also provide a fine tuning capability which will allow playing music in Pythagorean, just, meantone, or equal tempered scales with more than 12 notes per octave.

**Portamento is a continuous slide in pitch.*

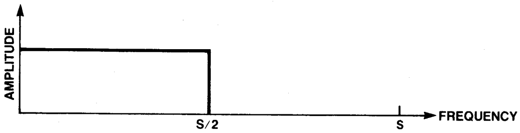


Figure 5a. Frequency Response of an Ideal Low Pass Filter

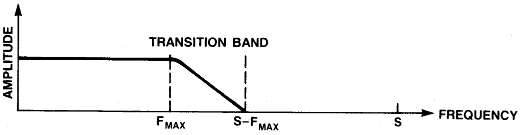


Figure 5b. Frequency Response of a Realizable Low Pass Filter

Waveform Memory

The size of the waveform memory will determine the amount of distortion or error noise in a single waveform (such as single sine wave). In FM synthesis the waveform memory is used to modulate the frequency of another oscillator. Thus the error in the waveform will be more noticeable in FM synthesis, especially for low frequencies. A 16384 word x 14 bits/word waveform (full cycle) RAM is used in a recently constructed [Alles and di Giugno 1977] high speed digital music module at Bell Research Lab (Murray Hill) which has generated high fidelity music. The waveform memory should be fast (have a low read access time) since it will be time multiplexed to generate a large quantity of partials.

Different schemes have been used to reduce the size of the waveform memory. Interpolation of the memory output data is extremely effective in increasing the signal to noise ratio of a small waveform memory [Moore 1977]. However, interpolation requires a multiplication and thus has not been used for real-time synthesis. (Multiplication is more expensive than memory as of 1977). A simple method of halving the required amount of waveform memory is to round off the phase angle to 13 bits, instead of truncating to 14 bits, in forming the waveform memory address.

By observing the symmetry of the sine function, one may realize that only ¼ of the amount of memory required for a full sine cycle is really needed. In other words, a continuous full cycle sine wave may be generated from the first quarter cycle (0° to 90°) of a sine function stored in PROMs. If different constant waveshapes are desired, then RAM should be used to store a full cycle of the waveform. A few of the problems which arise from attempting to generate interesting timbres with constant non-sinusoidal waveforms were discussed in the section entitled "Timbre". RAM waveform memory is more flexible, yet presently more expensive and more wire wrap or PC board space consuming (since a full cycle of an asymmetrical waveform must be stored) than the PROM memory needed to store ¼ of a sine cycle. Since Monolithic Memories, Inc. manufactures a large PROM (63RA841), the implementation of a ¼ sine cycle will be described here. Three of these 2048 word x 4 bit 63RA841 PROMs may be used to store the quarter cycle sine table. This 2048 word x 12 bits/word quarter sine table (with round off addressing) will have a signal to noise ratio equivalent to that of a 16384 word x 13 bits/word full cycle sine table with truncation addressing.

It may be extrapolated from the figures in "Table Lookup Noise for Sinusoidal Digital Oscillators" [Moore 1977] that a 16384 word x 13 bits/word full cycle sine table with truncation addressing would result in a worst case signal to noise ratio (S/N) of approximately 71.7 dB. The resulting total system signal to noise ratio would be a maximum of approximately 85 dB for music made with many voices. This is due to the digital to analog conversion system which is limited by the *sample and hold* or *track or ground switch* (used to deglitch the DAC output). However, an 85 dB S/N is as good as the S/N of analog synthesizers and much better than the S/N of audio tape recorders. If a wider dynamic range is desired, a floating point DAC system [Lee and Lipshutz 1976] may be used somewhat like the dbx units used to boost the dynamic range of audio tape.

There are currently several single IC quarter sine ROM s available. However only the MMI 6085 ROM has an access time around 100 ns and contains 1024 words. Using round off addressing, MMI's 1024 word x 10 bits/word quarter sine ROM will result in a signal to error noise ratio of approximately 64 dB [Moore 1977].

A simple method of implementing a quarter sine table with a minimal number of IC's is shown in Figure 6. Since sin(180° + α) = $-\sin(\alpha)$, a full cycle may be formed from the positive half of the sine cycle by forming the 2's complement of the PROM output data when the 180° bit (MSB) of the phase angle is high. The 180° bit is not used to address the PROM. The complement of this 180° bit controls a function select pin, S1, of the ALU which inverts the input when S1 is low. The ALU carry-in is fed by the 180° bit; thus the 2's complement of the sine half cycle is formed whenever the 180° bit is high. Three 74LS382 s are used to form the ALU because of their low heat generation, their ability to form the 2's complement, their relatively high speed without a carry look ahead generator IC, and the fact that they are packaged in the standard 0.3 inch row spacing DIPs. Throughout the oscillator it is desirable to: minimize concentrated heat generation from any one IC (fans are acoustically noisy in a music system with an S/N of approximately 85 dB); minimize the number of IC's, and minimize the propagation delay times of calculations (so that parts may be time multiplexed to generate many partials).

It may be remembered that sin(180° − α) = sin(α). Thus the PROM address of a quarter cycle sine table must be inverted when the phase angle is incremented through the second and fourth quadrants. When the 90° bit of the phase angle is high, the PROM address will be inverted using XOR gates as shown in Figure 6. The 2048 word x 12 bits/word PROM will contain sine values which are offset by ½ LSB of the PROM address [James 1977, Clapp 1977]. Thus sin (θ + 0.022°) is stored at PROM address θ; where θ is the phase angle (with its two most significant bits, 180° and 90°, removed) which is inverted (1's complement) by the 90° bit and XOR gates. This offset sine table accounts for taking the 1's complement (instead of the 2's complement) of the PROM address. The resulting S/N should be approximately equivalent to that which results from rounding off the phase angle to address a sine table (of the same size but without offset sine values), since both methods have a maximum angle error of ±½ LSB.

The multiplier is used to vary the amplitude of a sine wave as a function of time. Amplitude envelopes, FM index envelopes, and frequency envelopes [Moorer 1976] are essential for generating natural timbre. Envelopes may be generated by straight line segment approximations to the envelope curve. If infinitesimal line segments were used, any curve could be generated. When first turned on, the controller initializes the phase angle increment register and amplitude register. This is accomplished by tristating the registers following the envelope adders at the same time that the module bus buffer registers are enabled. Then the phase angle register and amplitude register acquire their initial values. The module bus buffer registers are then tristated and the registers following the envelope adders are enabled. This is their normal operating state.

The slope of the envelope at the beginning of a line segment is loaded into a slope register (through its module bus buffer register) by the controller. This slope is repeatedly added to the current amplitude (or frequency for frequency envelopes) while the result is repeatedly stored in the current amplitude (frequency) register until the slope of the envelope changes. Then a new envelope slope is loaded into the slope register. A 3½ minute crescendo or portamento would be possible with a 24 bit envelope generated at a 40 kHz sample rate. This would not provide slow enough ramps for some experimental music so 28 bits are used. This envelope generation technique is relatively simple minded. It would be better to provide for exponential envelopes in hardware instead of using the software so often to change the slope. The slope values are scaled (multiplied or divided) by the microprocessor in order to expand or shrink the time axis or the amplitude (frequency) axis of an envelope. The microprocessor may use a medium speed peripheral multiplier such as the 67516. With this method of envelope generation, different envelopes may be used for every individual note and the partials of a note may have different envelopes.

Generation of a Large Number of Sinusoidal Components

Building a separate oscillator for each sinusoidal component would be expensive if one desired over 100 sinusoidal components. However one low distortion digital oscillator may be time multiplexed to generate a large quantity of low distortion sinusoidal components (with independent control of each frequency and independent control of each amplitude). The number of sinusoidal components which the oscillator will generate is determined by the speed of the ICs and the sample rate.

If the sample rate is set to 32768 samples/sec, a new sample must be generated every 30.52 μs. Each output sample from the DAC is the sum of a sample from each of the sinusoidal components. If 200 sinusoidal components are desired, each sinusoidal component will be allotted 30.52 μs/200 = 152 ns to compute its sample. In our design with the use of a 7299270 Hz clock and a 32768 Hz sample rate we have 222 sinusoidal components.

What kind of modifications must be made to the digital oscillator in order to generate many sinusoidal components? In Figure 6, the phase angle increment register must be replaced by a RAM which contains the phase angle increment (frequency) of each

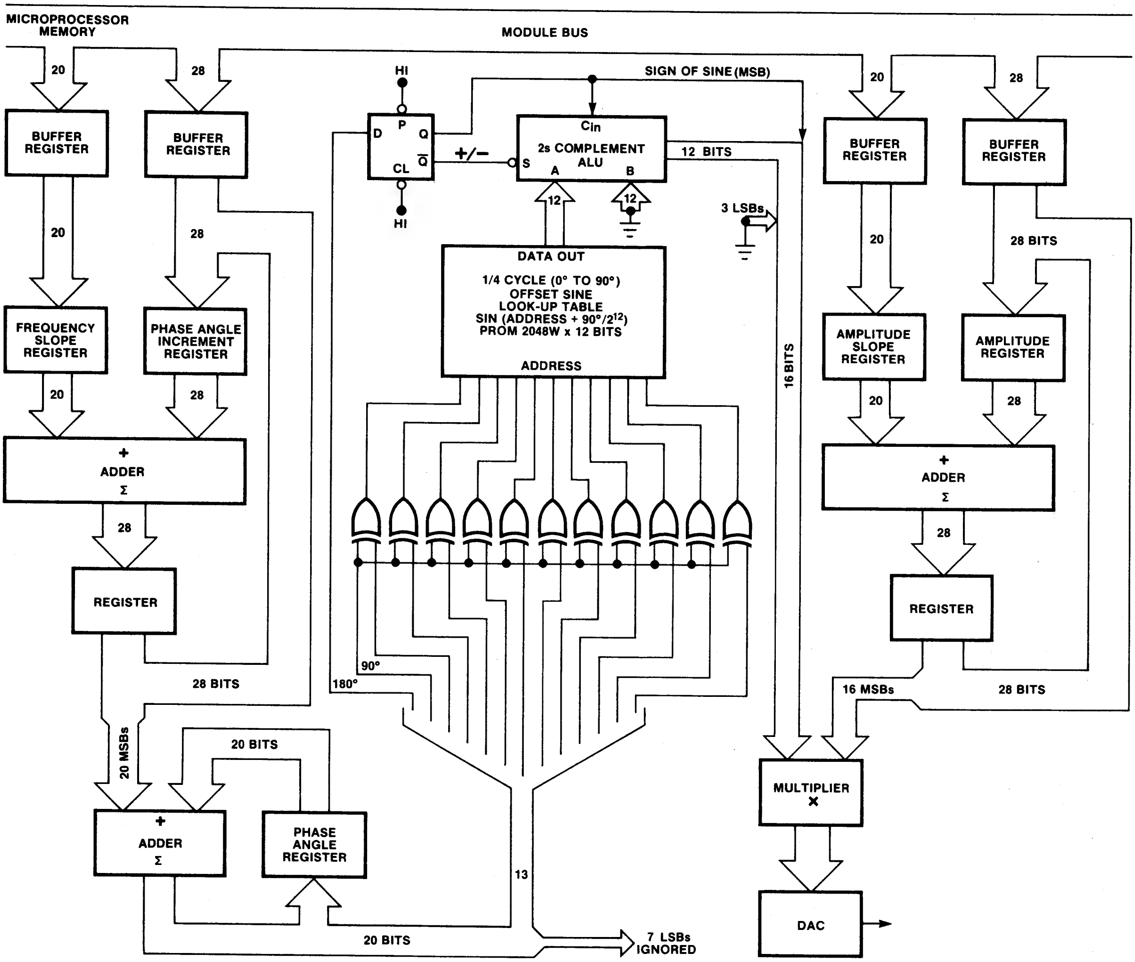


Figure 6. Basic Design of One Sinusoidal Oscillator With Frequency and Amplitude Envelopes. The XOR gates and offset sine table provide a simple method of addressing ¼ cycle of the sine function while generating a minimal amount of error noise [James 1977, Clapp 1977].

sinusoidal component. As shown in Figure 7, a phase angle RAM is used to hold the phase angle of each sinusoidal component. The frequency slope register, amplitude register, and amplitude slope register are likewise replaced by RAMs. Following the multiplier are two accumulators which are used to sum the sinusoidal components and voices. Counters are used to address the RAMs to determine which sinusoidal component is to be accessed.

If one had to perform all of the calculations for one sinusoidal component before starting the calculations for another sinusoidal component, one would not be able to calculate many sinusoidal components during the sample period. The pipeline registers placed after each operation allow many operations to take place simultaneously on different parts of different sinusoidal components. The individual component samples cannot move from one stage to the next before the slowest stage has finished its operation. Then all of the stages simultaneously move their contents to the following stage, and each accepts new data from its input. Many more sinusoidal components can be generated in real time since the pipeline registers increase considerably the throughput rate. In music, as in many signal processing applications, a stream of data is being processed. It is more important to maximize the throughput rate than to minimize the processing time for one individual piece of data.

The pipeline registers in Figure 7 allow a high throughput rate without having to use a large number of Schottky ICs. Unfortunately, there are a few bottlenecks in the pipeline where a small amount of Schottky logic is required. The multiplier is one bottleneck. One method of obtaining a fast multiplication throughput rate is with a parallel pipeline multiplier made with MMI's 74S558 integrated circuits as shown in Figure 8. Each of the 74S558 eight bit multiplier slices are immediately preceded by a Schottky pipeline register, thus minimizing the register clock to output time delay for this stage. Schottky pipeline registers also immediately follow the 74S558 multiplier slices, thus minimizing the register data setup time delay for this pipeline stage. The time delay for the multiplier pipeline stage will be the input register clock to output propagation delay time (17 ns max @ 25°C) plus the multiplier slice (74S558) delay time from input to output (60 ns max @ 0° to 75°C) plus the following register (74S374) data setup time (5 ns max @ 0°C to 70°C) before the clock's rising edge. The total delay time is thus around 82 ns for the multiplier stage of the pipeline. TRW's presently available (1977) MPY-16AJ has a definite advantage of being in one IC; however, it is clearly much slower. At 25°C, the MPY-16AJ multiplication time is a maximum of 190 ns. Since this IC draws approximately one amp of power supply current, it is not probable that it will remain cool. At 70°C, the multiply time should be a maximum of 217 ns [190 ns + (70°C − 25°C) × (0.6 ns/°C) = 217 ns]. Thus for speed and heat reasons a parallel pipeline multiplier (which outputs a new product every 82 ns) using the 74S558 integrated circuits is implemented here. The pipeline registers in Figure 8 allow the partial products to be added (following the 74S558s) with no extra delay time (since the previous partial products are being added at the same time that two new inputs are being multiplied).

Another bottleneck in the pipeline is a RAM stage. As the oscillator timing diagram in Figure 9 shows data must be written into these 93422 RAM ICs from the preceding pipeline register, the address changed, data read from the RAM and clocked into the following pipeline register all within one clock period. This is the slowest stage in our system flow and this delay determines the clock frequency. From Figure 9 the total time delay in this stage is the sum of:

1 — The read address access time of the RAM (60 ns)
2 — The data setup time of a register (5 ns)
3 — The clock to output of a register (17 ns)
4 — The data setup time (before write pulse) of the RAM (5 ns)
5 — The RAM write pulse width (45 ns)
6 — The RAM address hold time (5 ns)

This covers a total of 137 ns for the clock period.

In the accumulators which follow the multiplier, 74LS181 ALUs are used with 74S182 look ahead carry generators. The 74LS181s offer many extra useful functions, such as A + 0, which is needed at the beginning of a sample period. Alternatively, tristate registers with a clear input such as the 8T10 could be used with 74LS381s.

The first accumulator may be used to hold the sum of voices (instruments), while the second accumulator holds the components of one voice. In additive synthesis, 16 enveloped partials may be added together in the second accumulator. This sum would be input to the multiplier through the multiplexer where an overall amplitude envelope would be applied to finish forming the voice. This voice would then be added to the other voices in the first accumulator. In FM synthesis, several modulators may be added together in the second accumulator and this sum would be clocked into the FM register to modulate the frequency of the carrier(s). The resulting FM voice would be amplitude enveloped in the multiplier and added to other voices in the first accumulator. The FM register should be cleared when not being used.

A pseudo random number generator is used as a noise source. It is composed of a 24 stage shift register with feedback through XOR gates. Bits 24 and 23 are XORed together while bits 22 and 17 are being XORed. These two results are next XORed and fed to the serial input of the shift register. See *Electronotes* for details [Hutchins 1977]. This noise may be added to voices through the ALU or to the frequencies of components through the FM register. This multiplication-accumulation stage may be made into a general purpose signal processor by replacing the accumulating registers by pipelined RAMs and providing for feedback from the ALU to the other side of the multiplier [H. Alles and P. di Guigno 1977]. This would allow digital filtering (with filter coefficients stored in the amplitude envelope RAMs), plus ring modulation as well as a large variety of other functions. A path back to the microcomputer, which provides control parameters (frequencies and amplitudes), is provided for further non-real-time processing.

This module is controlled by a microprogram controller such as the 67110. The microprogram controller addresses a RAM every 137 ns to read a source address, a destination address, and control lines (encoded) as suggested by P. di Giugno. The sources and destinations may be a microcomputer's memory and modules such as the oscillator described in this article.

Synchronous Schottky down-counters (74S169s) are used to hold and *quickly* change the address to the RAMs, thus matching the 137 ns pipeline clock period. Everytime the amplitude slope RAM's counter (or frequency slope RAM's counter) counts down to 0, it will hold that address for one period (like any other period), then synchronously load the number of sinusoidal components to be generated −1 into the counter on the rising edge of the counter's clock. It will do this a second time on the next clock's rising edge. Thus after the end of its count down, the amplitude slope RAMs (or frequency slope RAMs) counter will skip one cycle. The other RAMs counter will *not* skip a cycle.

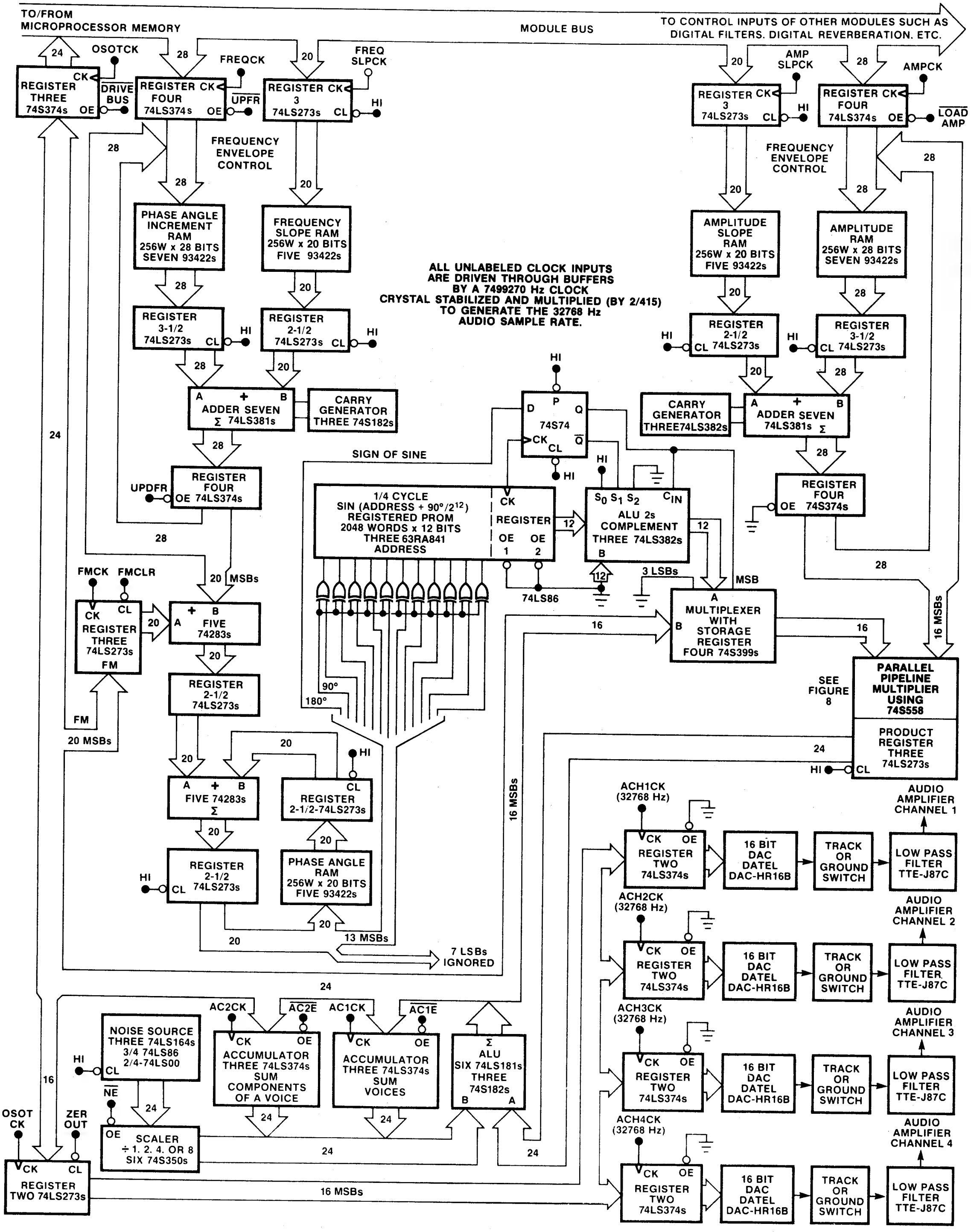

TO/FROM MICROPROCESSOR MEMORY
MODULE BUS
TO CONTROL INPUTS OF OTHER MODULES SUCH AS DIGITAL FILTERS. DIGITAL REVERBERATION. ETC.
24
OSOTCK
28
FREQCK
20
FREQ SLPCK
20
AMP SLPCK
28
AMPCK
REGISTER THREE 74S374s CK OE
DRIVE BUS
REGISTER FOUR 74LS374s CK OE
UPFR
REGISTER 3 74LS273s CK CL
HI
REGISTER 3 74LS273s CK CL
HI
REGISTER FOUR 74LS374s CK OE
LOAD AMP
28
FREQUENCY ENVELOPE CONTROL
FREQUENCY ENVELOPE CONTROL
28
28
20
20
28
PHASE ANGLE INCREMENT RAM 256W x 28 BITS SEVEN 93422s
FREQUENCY SLOPE RAM 256W x 20 BITS FIVE 93422s
AMPLITUDE SLOPE RAM 256W x 20 BITS FIVE 93422s
AMPLITUDE RAM 256W x 28 BITS SEVEN 93422s
ALL UNLABELED CLOCK INPUTS ARE DRIVEN THROUGH BUFFERS BY A 7499270 Hz CLOCK CRYSTAL STABILIZED AND MULTIPLIED (BY 2/415) TO GENERATE THE 32768 Hz AUDIO SAMPLE RATE.
28
20
20
28
REGISTER 3-1/2 74LS273s CL
HI
REGISTER 2-1/2 74LS273s CL
HI
HI
REGISTER 2-1/2 74LS273s CL
HI
REGISTER 3-1/2 74LS273s CL
28
20
HI
20
28
A + B ADDER SEVEN Σ 74LS381s
CARRY GENERATOR THREE 74S182s
D P Q 74S74 CK CL Q̄
CARRY GENERATOR THREE74LS382s
A + B ADDER SEVEN 74LS381s Σ
28
SIGN OF SINE
HI
HI
28
REGISTER FOUR OE 74LS374s
UPDFR
1/4 CYCLE SIN (ADDRESS + 90°/2¹²) REGISTERED PROM 2048 WORDS x 12 BITS THREE 63RA841 ADDRESS
CK REGISTER OE 1 OE 2
12
S0 S1 S2 CIN ALU 2s COMPLEMENT THREE 74LS382s B
12
REGISTER FOUR OE 74S374s
24
28
28
FMCK FMCLR
20 MSBs
12
3 LSBs
MSB
74LS86
CK CL REGISTER THREE 74LS273s FM
20
+ B A FIVE 74283s
20
A MULTIPLEXER WITH STORAGE REGISTER B FOUR 74S399s
16
16
16 MSBs
PARALLEL PIPELINE MULTIPLIER USING 74S558
SEE FIGURE 8
REGISTER 2-1/2 74LS273s
90°
180°
FM
20 MSBs
PRODUCT REGISTER THREE 74LS273s
24
HI CL
20
20
HI
A + B FIVE 74283s Σ
CL REGISTER 2-1/2-74LS273s
16 MSBs
ACH1CK (32768 Hz)
AUDIO AMPLIFIER CHANNEL 1
20
20
CK OE REGISTER TWO 74LS374s
16 BIT DAC DATEL DAC-HR16B
TRACK OR GROUND SWITCH
LOW PASS FILTER TTE-J87C
HI
REGISTER 2-1/2 CL 74LS273s
PHASE ANGLE RAM 256W x 20 BITS FIVE 93422s
ACH2CK (32768 Hz)
AUDIO AMPLIFIER CHANNEL 2
20
20
13 MSBs
7 LSBs IGNORED
CK OE REGISTER TWO 74LS374s
16 BIT DAC DATEL DAC-HR16B
TRACK OR GROUND SWITCH
LOW PASS FILTER TTE-J87C
24
AC2CK
AC2E
AC1CK
AC1E
ACH3CK (32768 Hz)
AUDIO AMPLIFIER CHANNEL 3
NOISE SOURCE THREE 74LS164s 3/4 74LS86 2/4-74LS00 CL
HI
CK OE ACCUMULATOR THREE 74LS374s SUM COMPONENTS OF A VOICE
CK OE ACCUMULATOR THREE 74LS374s SUM VOICES
Σ ALU SIX 74LS181s THREE 74S182s B A
CK OE REGISTER TWO 74LS374s
16 BIT DAC DATEL DAC-HR16B
TRACK OR GROUND SWITCH
LOW PASS FILTER TTE-J87C
16
NE
24
24
24
ACH4CK (32768 Hz)
AUDIO AMPLIFIER CHANNEL 4
OSOT CK
ZER OUT
OE SCALER ÷ 1. 2. 4. OR 8 SIX 74S350s
24
24
CK CL REGISTER TWO 74LS273s
16 MSBs
CK OE REGISTER TWO 74LS374s
16 BIT DAC DATEL DAC-HR16B
TRACK OR GROUND SWITCH
LOW PASS FILTER TTE-J87C

Figure 7. Real-Time Generator of up to 256 Sinusoidal Components.

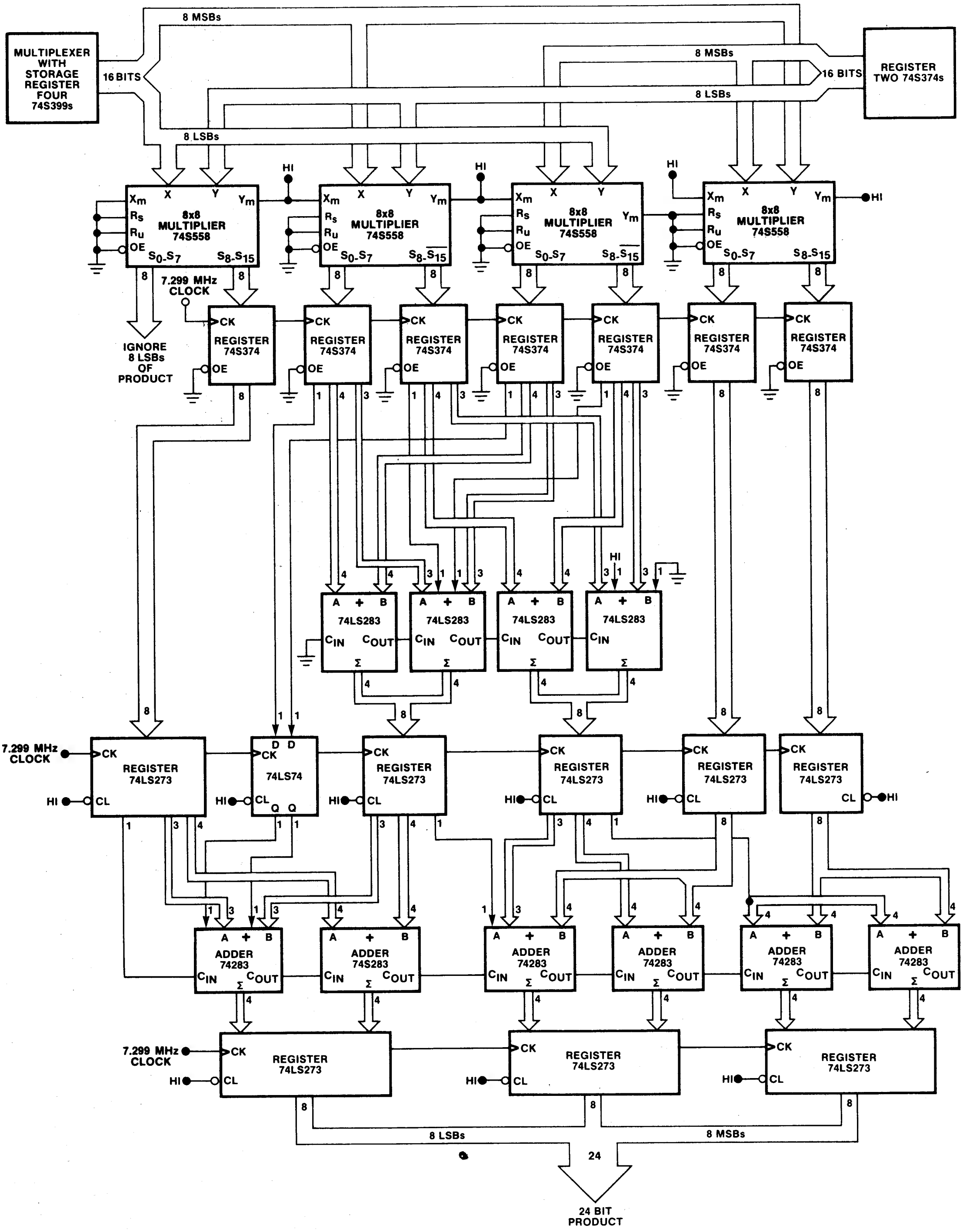


Figure 8. Parallel Pipeline 16 Bit x 16 Bit Multiplier. A new 24-bit 2's complement product will be output (registered) every 137 ns maximum.

This accounts for the data being written into these RAMs at one address after the address from which it was read before being incremented. The RAM address counters are clocked 67ns after the rising edge of the pipeline clock.

The frequency or amplitude slopes may be updated by loading the new value into the data register which precedes their RAM. The number of the sinusoidal component to be updated is loaded into a component number register. When the counter (which addresses the RAM) outputs this component number, the RAM is permitted to write the update value into the addressed location. The amplitude and phase angle increment RAMs are updated similarly, however, since these RAMs are written into every cycle, their normal source of data (the register following the envelope adder) must be tristated and the module bus buffer register enabled for the one update cycle. Then the module bus buffer register would be tristated and the normal source of data enabled.

It should be mentioned that the sine table is made from MMI's registered PROMs. These 63RA841 PROMs conveniently have a pipeline register built into them. Access time is specified as the internal register set up time which is typically 40 ns. Thus this pipeline stage including the XOR gates has no problem meeting the pipeline clock cycle time.

Digital to Analog

Datel manufactures a 16 bit DAC called the DAC-HR 16B which costs $299. Analog devices makes a high fidelity 8-bit DAC and a less expensive 16-bit DAC. A less expensive 12-bit DAC such as Harris Semiconductor's HI-562A-5 might be used initially and later replaced with a higher fidelity DAC. This Harris DAC will settle to ±1/10 LSB in 150 ns at 25°C and costs $29 in 100 quantity.

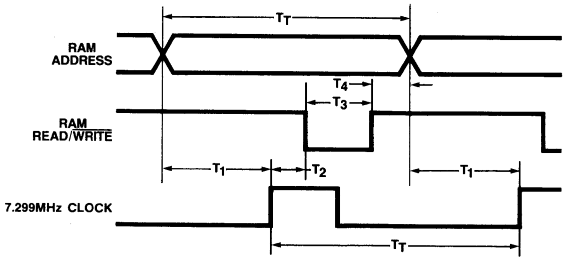
The track or ground switch should have an extremely high slew rate and introduce a minimal amount of sample period jitter. This switch should thus be made with bipolar transistors.

The low pass filter needs to be maximally flat in its pass band and have a very sharp frequency roll off above the cut off frequency. The J87C made by T. T. Electronics is a filter which fits this description fairly well. The J87C is a very low noise and low distortion filter (since it is passive) which has approximately ½dB of ripple. The transition bandwidth from the cut off frequency to 0 output is narrow enough to allow output frequencies of 42.5% of the sample rate.

Microcomputer Controlled Synthesizer Module

This oscillator may be thought of as one module which may be "plugged" into a modular digital synthesizer (which includes other modules such as digital reverberation, digital filters, analog to digital converters followed by a Fast Fourier Transform circuit, etc.) A microcomputer such as D.E.C.'s LSI-II is useful as a programmable interface between the module controller (high speed sequencer) and keyboards, joysticks, pressure sensitive pads, graphics terminals, etc..

It is hoped that this article will stimulate other engineers to design more flexible and less expensive computer architectures for the generation of high quality new music as rich as that of a large choir and orchestra or as soulful as the sound of a shakuhachi flute.



T_1 = the read address access time of the RAM plus the data set up time of a register.

T_2 = the clock to output delay of a register plus the data set up time (before the beginning of the write pulse) of the RAM.

T_3 = the RAM write pulse width.

T_4 = the RAM address hold time.

T_T = the total delay time of one pipeline stage = 137ns.

Figure 9. A Timing Diagram For the Real Time Sound Generator in Figure 7. The RAM address changes 67ns after the rising edge of the pipeline clock.

References

Alles, H. and P. di Giugno, "A One Card 64 Channel Digital Synthesizer," *Computer Music Journal,"* Vol 1, No. 4, 1977.

Backus, John, *The Acoustical Foundations of Music,* (W.W. Norton, New York, 1969) pp. 101-104.

Benade, Arthur, *Fundamentals of Musical Acoustics,* (Oxford University, New York, 1976). pp. 54-58.

Carlson, Bruce, *Communication Systems: An Introduction to Signals and Noise in Electrical Communication,* pp. 272-288 (McGraw-Hill, 1968).

Chowning, John, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", *Journal of the Audio Engineering Society,* Vol 21, No 7, 1973, pp. 526-534 (reprinted in *Computer Music Journal,* Vol 1, No 2, 1977).

Grey, John, *An Exploration of Musical Timbre,* Stanford University Dept. of Music Report No. STAN-M2; Feb., 1975.

Hutchins, Bernie, "The ENS-76 Home Built Synthesizer System —Part 8, Random Sources", *Electronotes-Newsletter of the Musical Engineering Group,* Vol 9 No 76, April, 1977.

James, Roger H. and David T. Clapp, letters (unpublished) to the editor of *Computer Music Journal,* 1977.

Le Brun, Mark, "Nonlinear Waveshaping", a paper presented at the International Computer Music Conference in 1977 at the University of Calif. at San Diego.

Lee, Francis and Lipschutz, David, "Floating Point Encoding for Transcription of High Fidelity Audio Signals", *Journal of the Audio Engineering Society* Vol 25, No 5, May, 1977.

Mathews, Max, with the collaboration of Joan E. Miller, F.R. Moore, J.R. Pierce and J.C. Risset, *The Technology of Computer Music,* (MIT Press, 1969).

Mathews, Max V. and F.R. Moore, "GROOVE—A Program to Compose, Store, and Edit Functions of Time," *Comm. of the ACM,* Vol 13, No. 12, December, 1970, pp 715-721.

Moore, Richard, "Table Lookup Noise for Sinusoidal Digital Oscillators", *Computer Music Journal,* Vol 1, No 2, 1977.

Moorer, James A., "Signal Processing Aspects of Computer Music—A Survey", *Computer Music Journal,* Vol 1, No 1, (1977) pp. 4-37.

Moorer, James A., "The Use of the Phase Vocoder in Computer Music Applications", presented at the 55th convention of the Audio Engineering Society, Oct 29 to Nov 1, 1976, Preprint 1146 (E-1).

Moorer, James A., "The Heterodyne Filter as a Tool for Analysis of Transient Waveforms", Stanford Artificial Intelligence Laboratory Memo 208, 1973.

Moorer, James A., *On the Segmentation and Analysis of Continuous Musical Sound by Digital Computer,* Ph.D. Thesis, Stanford University, 1975. Distributed as Dept. of Music Report No. STAN-M-3.

Moorer, Grey, and Snell, "Lexicon of Analyzed Tones—Part 1: A Violin Tone", *Computer Music Journal,* Vol 1, No 2, 1977.

Moorer, Grey, and Strawn, "Lexicon of Analyzed Tones—Part 2: Clarinet and Oboe Tones", *Computer Music Journal,* Vol 1, No 3, 1977.

Risset, Jean-Claude and Max V. Mathews, "Analysis of Musical Instrument Tones," *Physics Today.* Vol. 22, No. 2, 1969. pp 23-30.

Schottstaedt, Bill, "The Simulation of Natural Instrument Tones using Frequency Modulation with a Complex Modulating Wave", *Computer Music Journal,* Vol 1, No 4, 1977.

Snell, John, "Design of a Digital Oscillator which will Generate up to 256 Low Distortion Sine Waves in Real Time", *Computer Music Journal,* Vol 1, No 2, 1977.

Talambiras, Robert, "Some Considerations in the Design of Wide-Dynamic-Range Audio Digitizing Systems", Audio Engineering Society Preprint (1226 A-1), 56th Convention, May 10-13, 1977.

Warnock, Richard, "Longitudinal Digital Recording of Audio", Audio Engineering Society Preprint 1169(L-3), 55th Convention, October 29-November 1, 1976.

For more information on high quality musical applications of digital electronics, the Computer Music Journal is recommended, and is available from the People's Computer Co. (a non-profit corporation), P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94025.

For exchange of information about more versatile, high quality music instruments which are playable in real time, contact the author: John Snell (415) 323-3111, P.O. Box 279, Menlo Park, CA 94025.

Using ADPCM for Image Compression

Nadia Sachs

Digital communication is a rapidly growing area causing new concern for efficient transmission and storage of data, voice and video information. Coding and compression by reducing the bandwidth required for transmission and the memory needed for storage to decrease the system cost.

A simple method of compression Adaptive Differential Pulse Code Modulation (ADPCM) is explained and illustrated for video signals.

Using ADPCM For Image Compression

Introduction

Broadcasting, video conferencing services, facsimile transmission of newspaper, weather photographs and earth-resource pictures are just a few of the many applications which require image transmission. Prior to transmission the image is processed and coded. For digital channels, and to facilitate complex manipulations before transmission on an analog channel, the image processing is done in the digital domain.

To obtain a digital representation an image is scanned along one spatial coordinate in a line by line raster scanning process and the line scanned signal is time sampled (See Fig. 1). A discrete image may be regarded as a matrix of samples, X(I,J), referred to as pixels, which represent signal amplitudes.

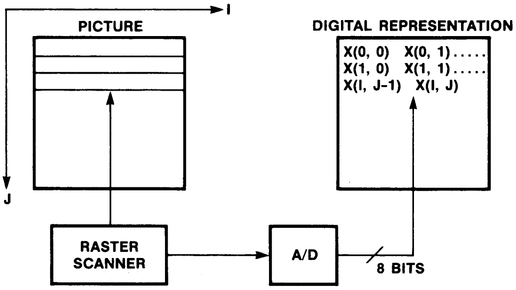


Fig. 1. To obtain a digital representation, an image is scanned horizontally and the signal is converted to 8-bit PCM samples X(I,J).

With conventional binary coding called "Pulse Code Modulation" (PCM) each of the pixels is coded with B number of bits; B=1 for binary facsimile, B=6 to 8 for monochrome image and single color component.

PCM transmission will require anywhere from 10^6 to 10^9 bits/s, depending on the application. For example, a television image of 512 by 512 pixels per frame at 8bits/pixel and 30 frames/s the rate is nearly $60x10^6$ bits/s. This high data rate can be reduced by using compression techniques. The storage or channel capacity savings obtained by compressing the data translate into a substantial cost reduction.

One approach to achieve compression is by exploiting redundancy in the normally highly correlated image source. DPCM (Differential PCM) is a simple scheme using this method.

DPCM - Differential PCM

Highly correlated image source means that — in an average sense — the signal does not change rapidly from sample to sample; therefore the difference between adjacent samples should have a lower variance than the signal itself, fewer bits are needed to send the difference than to send the new sample.

Let X(I,J) be the J-th pixel on the I-th line of the input raster (scanning in a horizontal fashion) and Y(I,J) the vectors of reconstructed pixels.

The basic differential system is shown in Fig. 2. E(I,J) is the prediction error, a measure of the accuracy of the prediction made. To achieve compression, fewer bits are allocated to E(I,J) than X(I,J). This function is performed by the quantizer illustrated later. The only information available at the receiver is the previously reconstructed signal, Y(I,J-1) and the prediction error E(I,J). Consider the transmitter and receiver equations:

$$E(I,J) = X(I,J) - X(I,J-1) \qquad \text{(for transmitter)}$$

$$Y(I,J) = E(I,J) + Y(I,J-1) \qquad \text{(for receiver)}$$

Note that any discrepancies (due to quantization noise) between input samples, X(I,J) and reconstructed samples Y(I,J) are accumulated.

To prevent this phenomenon Y(I,J-1), instead of X(I,J-1), is used in the transmitter, ie:

$$E(I,J) = X(I,J) - Y(I,J-1) \qquad \text{(for transmitter)}$$

Therefore reconstruction, computation of Y(I,J-1), is incorporated in the transmission function as well as in the receiver (see Fig. 3). In order to initialize the system Y(0,-1) could be zero or any convenient value.

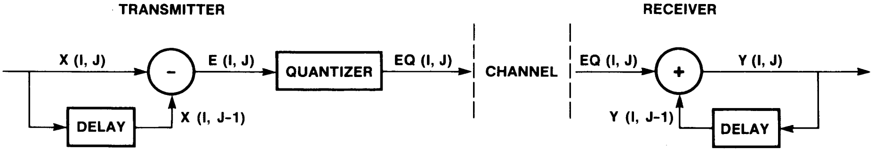


Fig. 2. Basic differential system: the difference between adjacent samples is quantized and transmitted.

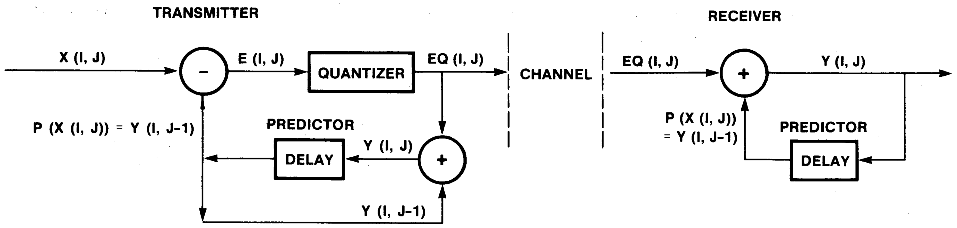


Fig. 3. DPCM system: for a new sample X(I,J), an estimate P(X(I,J)) is obtained from previous samples and EQ(I,J), the quantized prediction error, is transmitted.

Y(I,J-1) can be denoted as P(X(I,J)) since it represents the predicted value for X(I,J).

In summary: for a new sample X(I,J) an estimate P(X(I,J)) is obtained from previously reconstructed samples at both transmitter and receiver. The prediction error computed and quantized at the transmitter is sent over the channel and used at the receiver to regenerate the sample.

The two principal components of the DPCM system are the Predictor and the Quantizer.

Predictor

The role of the predictor is to estimate the value of the incoming sample, X(I,J), based on the previous data sample. Several predictor functions could be used for DPCM coding.

The one-dimensional predictor considers only pixels on the same row. In the simplest case described P(X(I,J)) = Y(I,J-1), however several samples could be used; for example:

$$P(X(I,J)) = \sum_{K=1}^{M} A(K) \cdot X(I-K,J) \qquad M \leq 4$$

A(K) the prediction coefficients are chosen to minimize the error between the predicted and the real value of the sample. Increasing the order of the predictor above four points does not give any appreciable improvement in performance.

The two-dimensional predictor function uses elements in the previous line as well; for example:

$$P(X(I,J)) = A1 \cdot Y(I-1,J) + A2 \cdot Y(I,J-1) - A3 \cdot Y(I-1,J-1)$$

Here close neighbors, from the adjacent column as well as from the previous line, are used.

Interframe predictors use picture elements also from the previously transmitted frame.

Predictors can be fixed or adaptive. The illustrated ones are fixed, being independent of the data. Adaptive predictors change in accordance with some decision rules operating on pixels in the local vicinity of the element to be predicted.

Since prediction of new samples is based on preceding samples, any transmission error in the receiver will accumulate and propagate.

The effect and spread of error depends on the predictor used. The previous element predictor: P(X(I,J)) = Y(I,J-1) can be stabilized by providing leak, ie. P(X(I,J)) = A•Y(I,J-1) where A≤1.

The effect of transmission error will decay; however, criteria for stability of any predictors are not well known.

Other alternatives to minimize transmission error include error detection and correction or insertion of an uncompressed line at regular intervals.

Quantizer

The quantizer will perform a mapping of E(I,J) into fewer bits. Starting with 8 bits for X(I,J),n (where n less than 4) bits can be allocated to EQ(I,J); n can be constant or vary from sample to sample. A fixed quantizer (n constant) has the advantage of maintaining a constant output bit rate.

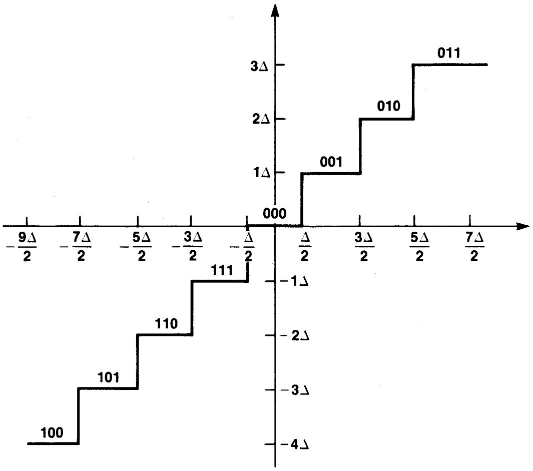


Fig. 4. Uniform quantizer characteristics (Δ is the quantizer step).

Various quantizer types are used for DPCM, some are designed on a statistical basis; others take a different approach such us psychovisual measures. The first considers the probability of an error, the second the "visibility" effect. The visibility function relates the magnitude of the prediction error at a pixel to subjective visibility effects when noise is added to that pixel.

One example: the uniform quantizer (see Fig. 4) is derived on a statistical basis criterion; it is optimal for uniformly distributed variables. A uniform quantizer EQ(I,J) will correspond to the n least significant bits of E(I,J). The samples of the image are not uniformly distributed, so the variance of the input signal, and subsequently the variance of E(I,J), fluctuates with changes in the image. One approach to accomodate amplitude variations is to precede the uniform quantizer by a time varying gain tending to limit the range of E(I,J) to n bits. The gain can be derived by computing and continuously updating the standard deviation, G(I,J), of the error samples.

$$\text{gain} = \frac{1}{G(I,J)}$$

Three types of reconstructed image degradation can be seen due to improper design of the quantizer.

Granular noise - visible when small prediction errors corresponding to flat areas are coarsely represented

Edge busyness - seen when prediction errors corresponding to gradual change in edges cause quantizer oscillation

Slope overload - appears with large prediction errors corresponding to high contrast edges if the largest quantizer level is too small

ADPCM Hardware Implementation

An example of an ADPCM (Adaptive DPCM) system is shown in Fig. 5 (transmitter only).

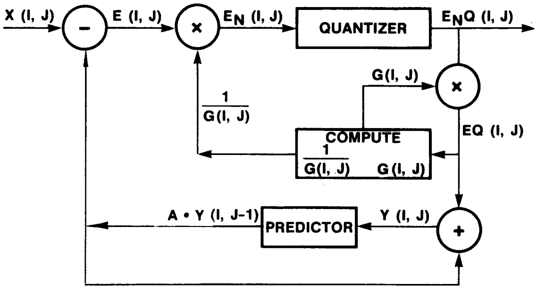


Fig. 5. ADPCM system: a DPCM system with an adaptive quantizer. The prediction error E(I,J) is normalized using an updated standard deviation followed by a fixed quantizer.

The predictor is a first order one dimensional predictor of the form:

$$P(X(I,J)) = A \bullet Y(I,J-1)$$

A is the attenuation constant that makes the predictor stable by minimizing the effect of transmission error.

The quantizer is an adaptive quantizer; the prediction error E(I,J) is normalized by using an updated standard deviation, followed by a n bit fixed quantizer. The value of n will determine the compression ratio and will affect the reconstructed image quality. Let G(I,J) be the variance of E(I,J). The estimate used for G(I,J) is based on the recursion:

$$G^2(I,J) = (1-h) \bullet EQ^2(I,J-1) + h \bullet G^2(I,J-1) \qquad h<1$$

called an exponential variance estimator.

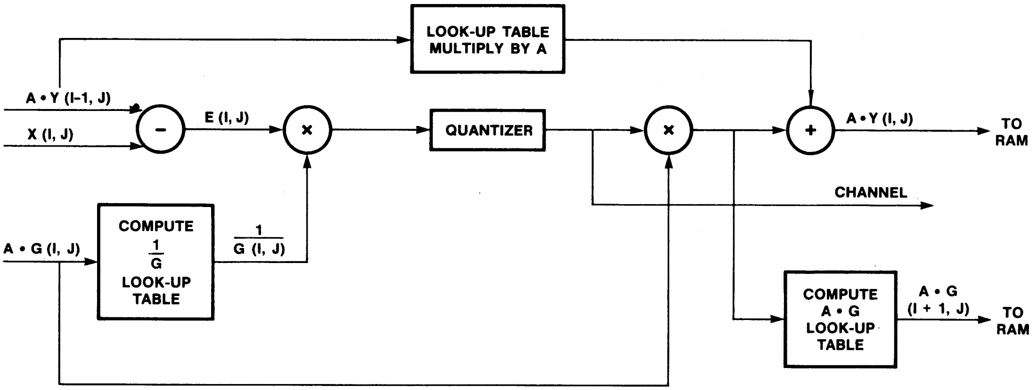


Fig. 6. The ADPCM system of Fig. 5 was rearranged for a suitable pipeline implementation.

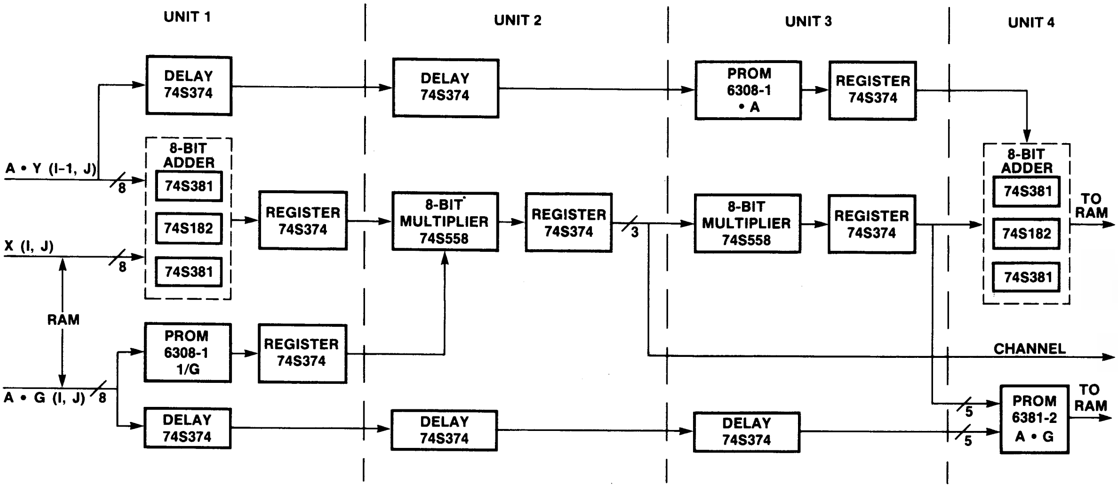


Fig. 7. Detailed implementation of the ADPCM system.

The feedback path prohibits a pipeline implementation suitable for high speed. If column information is used instead of prediction and adaptation on row information, the feedback paths are broken. The piped system is shown in Fig. 6. The predictor leak function is performed before the adder by premultiplying the adder inputs by A. The penalty for column computation is the need to store one row of reconstructed data and one row of variance estimates.

A detailed implementation for the transmitter is shown in Fig. 7.

All the multiplication by constants, squaring, square root and inversion are performed using bipolar PROM look-up tables.

The maximum allowable frequency of operation is limited by the multiplier performance. The worst case propagation delay of the MMI 74S558 (an 8 bit combinatorial multiplier) is 60ns, making it very attractive for this application as well as for other video signal processing. The multiplier stage (including register) will have an 82 ns worst case delay:

60ns + 17ns(clock to out delay) + 5ns(set-up) = 82ns

In each 100 ns cycle, two memory accesses are performed from the A•Y(I,J) RAM, as well as the G(I,J) RAM. Since similar access sequences are needed, the same address can be used for both memories. If the RAM access time is greater than 50ns, the even-odd combination of the read and write cycle permits the use of interleaving techniques to achieve the required throughput.

The system is initialized by clearing the RAMs. This is accomplished by storing zeroes in all locations (for a complete row count).

Conclusion

The ADPCM system illustrated can be used for image compression stand alone or as part of a hybrid coder. In a hybrid coder a group of pixels is first transformed using Discrete Cosine Transform and the transform coefficients are transmitted using ADPCM. For information on hybrid coders as well as various other compression techniques, see the references.

The circuit described operates in real time for rates up to 12.2 Mbits/s and its hardware implementation is relatively simple.

The maximum compression ratio achieved is dependent on the level of distortion acceptable. A factor of 2.6 to 1 compression preserving image quality can be expected.

References

1) Arun N. Nitravali and John O. Limb, "Picture Coding: A review", Proc. IEEE10, Vol 68, pp 366-406, March 1980.

2) A. K. Jain, "Image Data Compression : A Review", Proc. IEEE10, Vol.69, pp 349-389, March 1981.

Notes

Shadow Register Architecture Simplifies Digital Diagnosis

John Birkner, Vincent Coli, Frank Lee

Abstract

A series of new devices including register and PROMs with diagnostics now make it easier for system designers to include diagnostic circuitry in microprogrammed systems. When in the diagnostic mode, these devices allow for complete system controllability and observability with a minimum of additional hardware. Other schemes such as embedding diagnostic code in a digital system and LSSD (Level-Sensitive Scan Design) have been used in the past, but these techniques have their drawbacks. This new series of products as well as microprogrammed architectures using these products will be explored in this paper.

Shadow Register Architecture Simplifies Digital Diagnostics

John Birkner, Vincent Coli, Frank Lee

Introduction

Testing of digital systems has always been very costly. In order to decrease the cost, the following must be resolved:

1. The time needed to fix a system should be minimized;
2. The additional hardware needed for testing should also be minimized.

The solution to the first issue is to use the aid of a computer. The system under test is first loaded with test input data. The result is then unloaded and compared to the expected result to identify any defective part. This methodology is called signature or fingerprint analysis.

The second issue involves the digital hardware and the console or the diagnostic bus needed for the test data. A wide bus just for testing will take too much board space and will not be worthwhile. On the other hand, if only one data input is needed for testing, it will be very inexpensive to perform testing. This can be achieved by serializing the test data and result. One concept which does this is LSSD (Level-Sensitive Scan Design). LSSD has its drawback, and so a new concept called the shadow register diagnostics was developed. These concepts will be discussed later in this Application Note

Microprogram Control Store and Other Signal Paths

Microprogramming is the technique of using control programs stored in high speed memory, such as bipolar PROMs and static RAMs, to instruct a digital system to perform various functions. A typical microprogram control store architecture is given in figure 1. The microprogram sequencer generates the addresses for the microprogram memory which stores the control program. The microinstruction register assures that all bits change simultaneously after the clock pulse and allows the execution of one microinstruction while fetching the next microinstruction. Some bits are fed back to the sequencer while others are used for system control. Since this is the central control unit of a processor, this is a place that tests must be performed in case of an error.

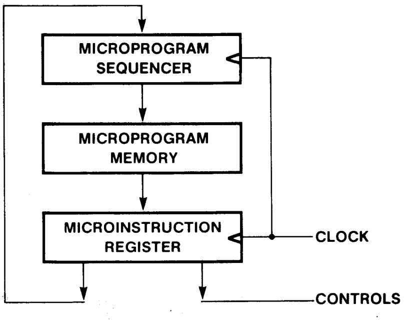


Figure 1. Typical Microprogram Control Store

Besides the microprogram memory, there are other signal paths which test should also be performed, namely data paths, address paths, and sequential logic. Diagnostic capability may also be required in these signal paths.

Diagnostic Testing

The basic theory of system diagnostics is to insert test data at the inputs of the system and to sample test results at certain nodes or the outputs. For a non-sequential circuit, testing is easier since the circuit has no memory of the previous states. But for a sequential circuit, the data to be sampled at a node depends not only on the inputs, but also on the previous state. If the previous state contains an error, it will possibly perform an illegal jump. In that case, depending on which state the system is currently in, the next state may be different. After several illegal jumps, it will be quite difficult to trace the error path.

One solution to this problem is to convert a sequential circuit into a non-sequential one when system testing is to be performed. A sequential circuit can often be viewed as a network with a clock and a number of inputs and outputs, with some outputs being fed back to the inputs (figure 2a). If the feedback path is broken and inputs which are fed back from the outputs are instead fed in from some external sources (figure 2b), the system becomes non-sequential, and system testing will be easier.

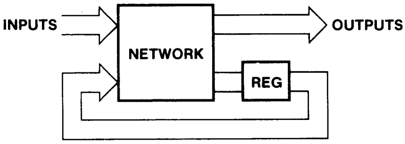


Figure 2a. A Sequential Network with Feedback

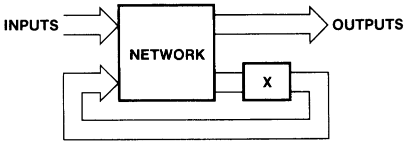


Figure 2b. Feedback Loop of the Network is Broken

Generation of test vectors can be achieved through commercially available simulation program.

Simple Microprogramming Architecture with Diagnostics

In a microprogramming environment, the next microinstruction is a function of the current microinstruction and status. In other words, the next state of the system depends on the current state and some control signals. Testing will be simpler if we break the feedback path from the microinstruction word to the microprogram sequencer.

One technique used in implementing system diagnostics is to expand the microprogram memory to accomodate the diagnostic words (figure 3). When performing diagnostic operations, the diagnostic words are loaded from the diagnostic locations of this memory. The advantage of this implementation is its simplicity — no additional hardware is needed. On the other hand, it has three disadvantages:

1. The memory space is limited. It is unusual to have a very deep microprogram storage (greater than 8K words). This limits the number of diagnostic words which can be stored. In addition, the number of test words are limited.
2. It does not provide a convenient method to sample or observe the test results and output them for analysis.
3. When there is an error, there may be an illegal jump and there will be no control over the next microcode address.
4. There may be some components such as the ALU which should be tested with other input data besides the microcode. This scheme may provide the microcode with diagnostic data but a question on how to test these parts still remains.

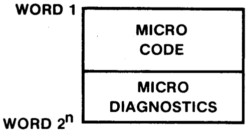


Figure 3. Microprogram Memory Map

An alternative method is to serially shift in diagnostic microinstruction data from an external source and shift the results out in a similar manner.

Shift Method Aids Testing

Diagnostic data can be shifted in from outside the system to the microinstruction register (figure 5). During normal operation, this register is loaded with the contents from the microprogram memory. When diagnostics are to be performed, the microinstruction register becomes a shift register and diagnostic data is shifted in serially. After a diagnostic microinstruction is completely shifted in, the system will run with the diagnostic data as though it is in normal operation. So a new microinstruction will appear on the outputs of the microprogram memory. This microinstruction is part of the test result. The microinstruction register will be loaded from the microprogram memory during the next clock cycle. Then during the next clock cycle, a new diagnostic microinstruction will be shifted in while the test result will be shifted out to be analyzed.

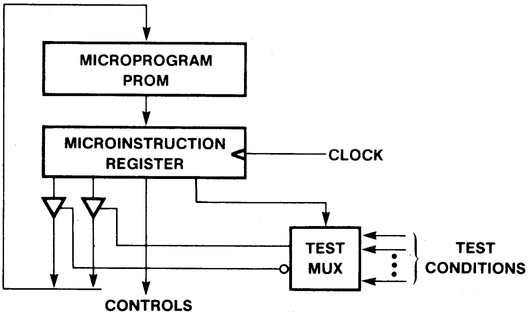


Figure 4. Simple Microcontroller

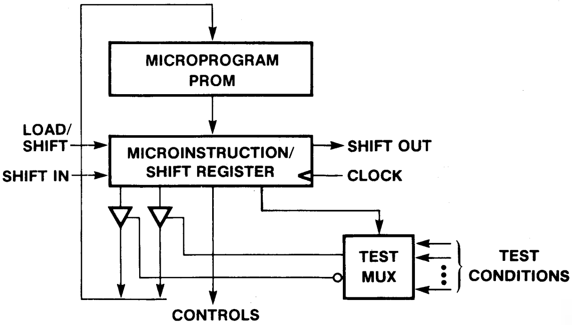


Figure 5. Microcontroller with Shift Register

One concept which uses the shift method is the Level-Sensitive Scan Design (LSSD). Outputs from the microinstruction latch will contain some intermediate data when diagnostic microinstructions are shifted in and test results are shifted out. If several control signals are used to drive several ports on the same bus, it is possible that more than one port may be enabled at the same time by the intermediate data (as shown in figure 6), thus creating a bus fight. the result may be hazardous to your system. Other hazards such as a disk crash are also possible. Designing with LSSD forces compromises in system design.

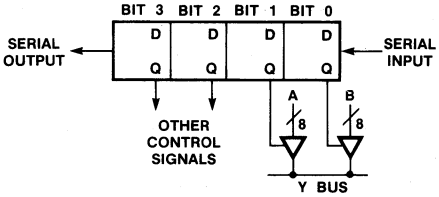


Figure 6. Potential Bus Fight May Appear as Port A, and Port B May Both be Enabled When Test Data is Shifted Through the Register Bits

If the diagnostic data is shifted into some buried registers which are not directly tied to the control lines, the above problem can be avoided. This is the concept of shadow register diagnostics.

Shadow Register Eliminates Hazards

A shadow register is basically a buried register with shift capability (figure 7). There is also an output register whose outputs appear to the rest of the system. Each output flip-flop has an associated flip-flop in the shadow register. For all diagnostic parts except for the 53/63DA1643, an output flip-flop drives a three-state buffer before going to the output pin. If the output is disabled, the output pin may be converted to an input pin. This feature is very important if the output is driving a bus and sampling of data on the bus is desired.

The input to a bit of the shadow register ia a multiplexer which can select data from one of three sources:

1. The less significant bit location in the shadow register (or SDI for the least significant bit). This operation is just a simple shift register.
2. The same bit location in the shadow register.
3. Data on the output pin at the same bit position. This data may be the output of the corresponding bit of the output register if there is no output enable pin or if the output is

enabled, or the input to that pin if there is an output enable pin and the output is disabled.

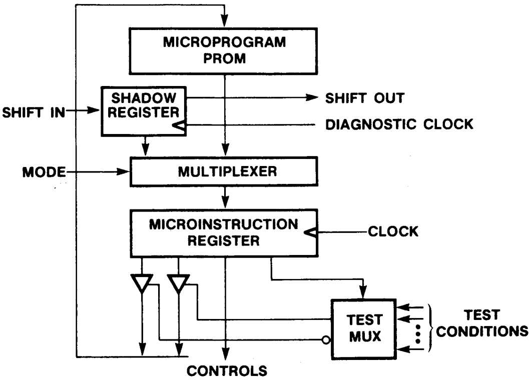


Figure 7. Microcontroller with Shadow Register

The input to any bit of the output register is also multiplexed from one of two sources:

1. The corresponding input bit from the memory array.
2. The corresponding bit location in the shadow register.

Since the data shifted in during the diagnostic mode does not appear on the output bus, the system will never see the intermediate results of the serial shift. Therefore, all control signals will be valid and the hazards associated with LSSD are eliminated.

With this concept in mind, a new standard for upcoming system diagnostics can now be presented.

A block diagram of a typical diagnostic component using the shadow register concept is given in figure 8.

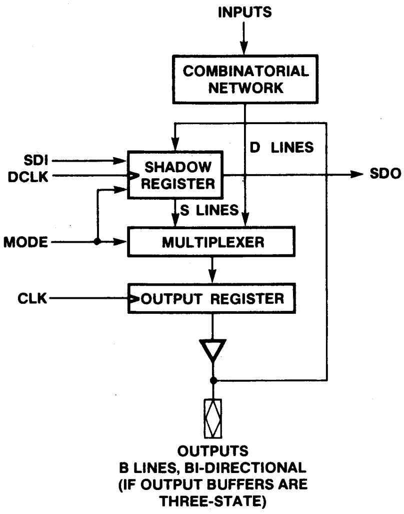


Figure 8. A Typical Diagnostic Part Using the Concept of the Shadow Register

Product with Diagnostic Capability

A family comprising of 4K, 8K, and 16K Diagnostic PROMs with 4-bit output organizations and 8-bit Diagnostic Register are being developed by Monolithic Memories. These devices are packaged in industry standard 24-pin SKINNYDIP™ (0.3 inch wide) packages and are specified over both commercial and military temperature ranges.

Features

Edge Triggered Registers — Data from the PROM, input data bus, or shadow register is loaded into the output register (Q_3- Q_0 for Diagnostic PROMs and B_7-B_0 for Diagnostic Register) on the rising edge of the clock (CLK). Similarily, data is loaded or shifted in the shadow register (S_3-S_0 for the Diagnostic PROM or S_7-S_0 for the Diagnostic Register) on the rising edge of the Diagnostic clock (DCLK). The advantage of using a register is that system timing is simplified, and faster microcycle times can be obtained.

Programmable Initialization — This feature appeared in the Diagnostic PROM family. The Output Register can be asynchronously loaded with a user-programmable initialization word. Each flip-flop in the Output Register may be individually programmed to either a High state or a Low state so that when the Initialize pin ($\overline{\text{IA}}$) is active, the output register will contain this initialization word. Note that the initialization operation can occur independent of a clock pulse. Also, this feature is a superset of a preset and clear function. Therefore Programmable Initialization can be used to generate any arbitrary microinstruction for system Reset or Interrupt.

Synchronous and Asynchronous Enables — Both synchronous and asynchronous output enable options are available in the Diagnostic PROM family. The synchronous enable ($\overline{\text{ES}}$, see figure 9a) is used when more than one Diagnostic PROM is bussed together to increase word length. In this case, the enables effectively become the most significant address bits and, as such, must be registered just as is the data. Stated another way, when the clock goes high, the address is free to change, requiring enable information to be remembered somewhere. It is most appropriate to store the enable information. When the enable is not used, or when the outputs are to be gated onto some type of bus, the registered enable tends to get in the way. For this reason, the asynchronous option ($\overline{\text{E}}$, see figure 9b) is offered to allow direct control of the enable. For parts which have both synchronous and asynchronous enables, outputs are enabled if and only if $\overline{\text{E}}$ is low and $\overline{\text{ES}}$ is LOW the last time clock goes HIGH (figure 9c). For the diagnostic register, there is only asynchronous output enable.

Three-State Drivers — For all components of the diagnostic family except for 53/63DA1643, the output of the register is buffered by three-state drivers which are compatible with low-power Schottky three-state bus standards. The 53/63DA1643 has totem-pole outputs.

Write-Back Inputs — The Diagnostic Register is to be used for data registers, address registers, instruction registers, or other registers. But it can also be used in the microinstruction register of the microprogram memory, in case a writable control store (WCS) is used. The diagnostic register can change the inputs to outputs to initialize the WCS.

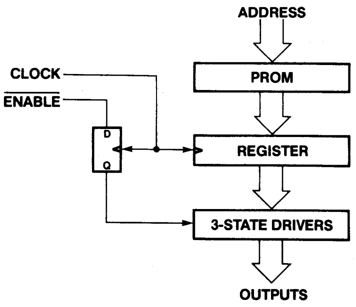


Figure 9a. Synchronous Output Enable

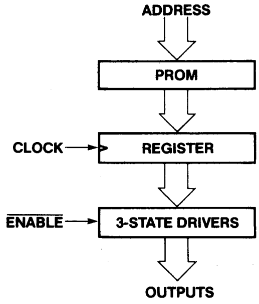


Figure 9b. Asynchronous Output Enable

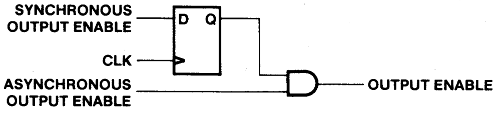
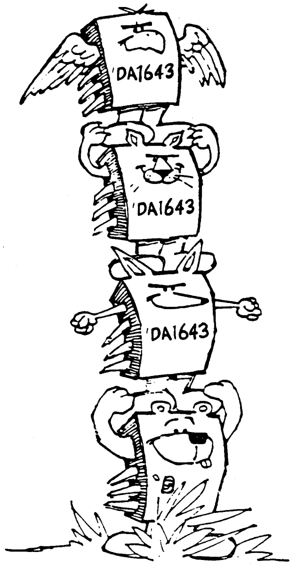


Figure 9c. Enabling of Outputs with Both Synchronous and Asynchronous Output Enables



"...THE 'DA1643 TEATURES ASYNCHRONOUS INITIALIZATION AND 'TOTEM-POLE' OUTPUTS..."

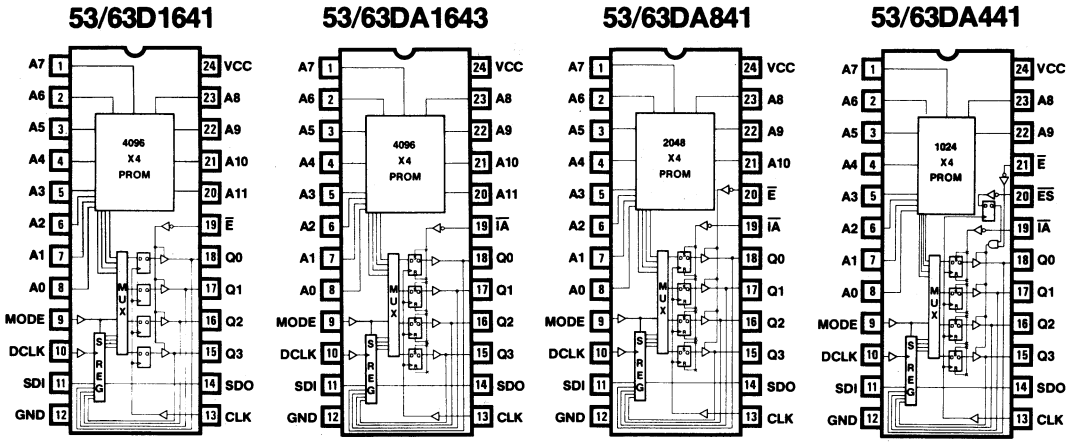


Figure 10. Diagnostic PROM Family Logic Symbols

Product Description

The Diagnostic component series consists of the following products (see figure 10 and 11 for the Logic Symbols):

53/63D1641 — 4096 words x 4 bit memory with asynchronous three-state enable.

53/63DA1643 — 4096 words x 4 bit memory with asynchronous initialization and totem-pole outputs.

53/63DA841 — 2048 words x 4 bit memory with asynchronous initialization and asynchronous three-state enable

53/63DA441 — 1024 words x 4 bit memory with asynchronous initialization and both asynchronous and synchronous three-state enables.

54/74S818 — 8-bit register with asynchronous three-state enable and write-back capability to the inputs, basically for writable control store (WCS) loading.

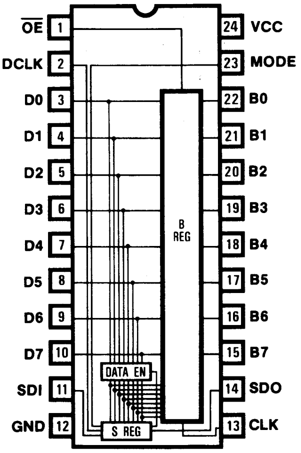


Figure 11. Diagnostic Register Block Diagram

Technology

The Diagnostic PROM and Register family utilizes Monolithic Memories advanced self-aligned washed emmitter bipolar process. PNP inputs provide high impedance inputs (0.25mA max) to the devices. In addition, the DPROMs utilize the Ti-W fusible link technology in an NPN emitter follower array to provide high speed and reliable devices.

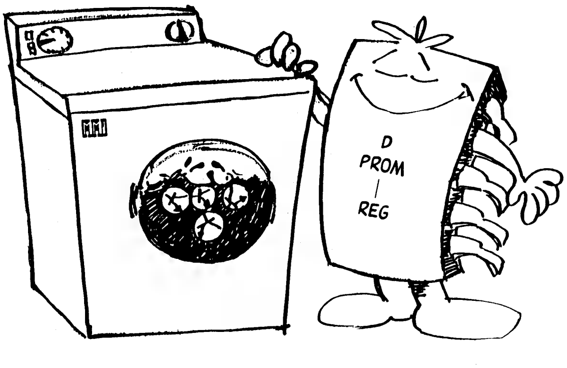
A Standard for the Diagnostic Parts

The introduction of the DPROMs and register with diagnostic results in a new standard for diagnostics. Noting that the Diagnostic devices need controls over two independent registers and multiplexer, a number of overhead pins are necessary. These overhead pins must be defined in a way that the diagnostic parts can be cascadable.

At a minimum, the following pins are needed in addition to those used in a similar part without the diagnostic features:

1. Diagnostic Clock (DCLK) — The diagnostic clock is used to clock the shadow register.
2. MODE — This pin is used in selecting the data to the registers. For the output register, MODE = LOW indicates that the output register is being used as a normal register; MODE = HIGH indicates that the next state of the output register will be obtained from the shadow register. For the shadow register, MODE = LOW indicates serial data from SDI (see below) is shifted in every diagnostic clock; MODE = HIGH switches SDI from a data input to a control input. See below for details.
3. Serial Data In (SDI) — When MODE = LOW, this pin is used for shifting serial data in. When MODE = HIGH, SDI serves as a control pin. If MODE = HIGH and SDI = LOW, data from the output pins will be loaded to the shadow register on the next DCLK. MODE = HIGH and SDI = HIGH indicate a reserved operation for diagnostic PROMs, and is used for write-back for diagnostic register.
4. Serial Data Out (SDO) — When MODE = LOW, this pin carries the shift-out bit of the shadow register. When MODE = HIGH, the SDI becomes a control pin and the control signal should be passed along if several diagnostic parts are connected together serially. So SDO should carry SDI along in this case.

This standard is being used in designing all current and future diagnostic devices.



"...THE DIAGNOSTIC PROM AND REGISTER FAMILY UTILIZES MONOLITHIC MEMORIES ADVANCED SELF-ALIGNED WASHED EMITTER BIPOLAR PROCESS..."

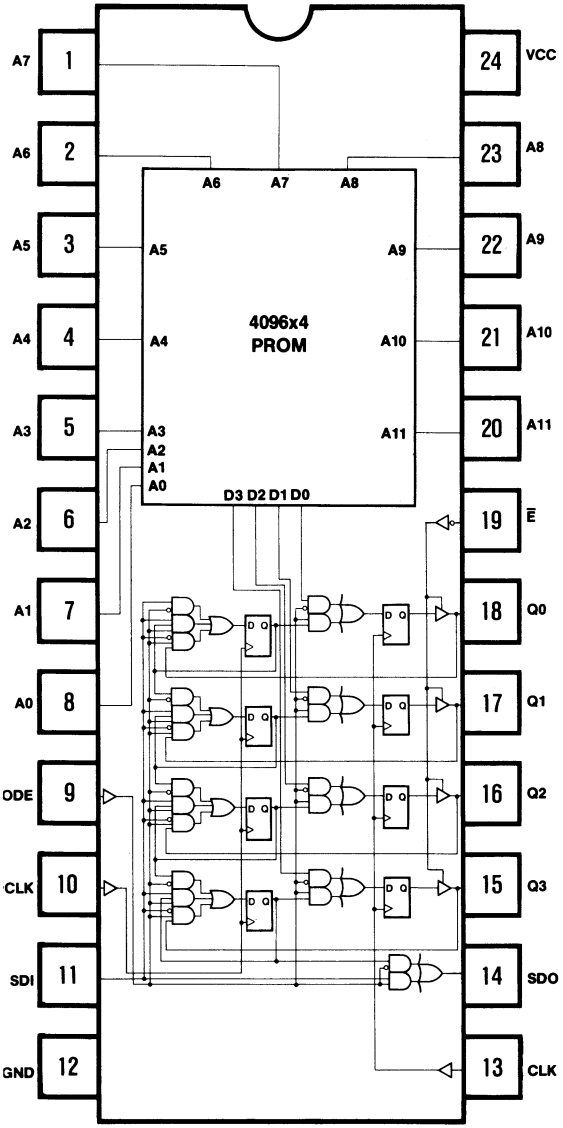


Figure 12a. 53/63D1641 4096x4 Diagnostic PROM with Three-State Outputs

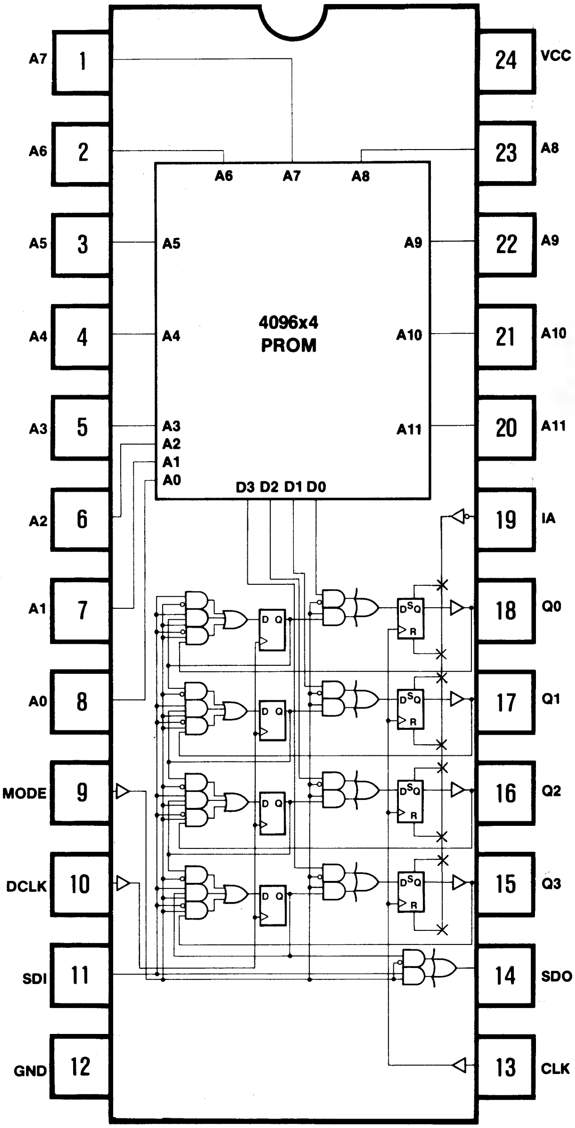


Figure 12b. 53/63DA1643 4096x4 Diagnostic PROM with Asynchronous Initialization

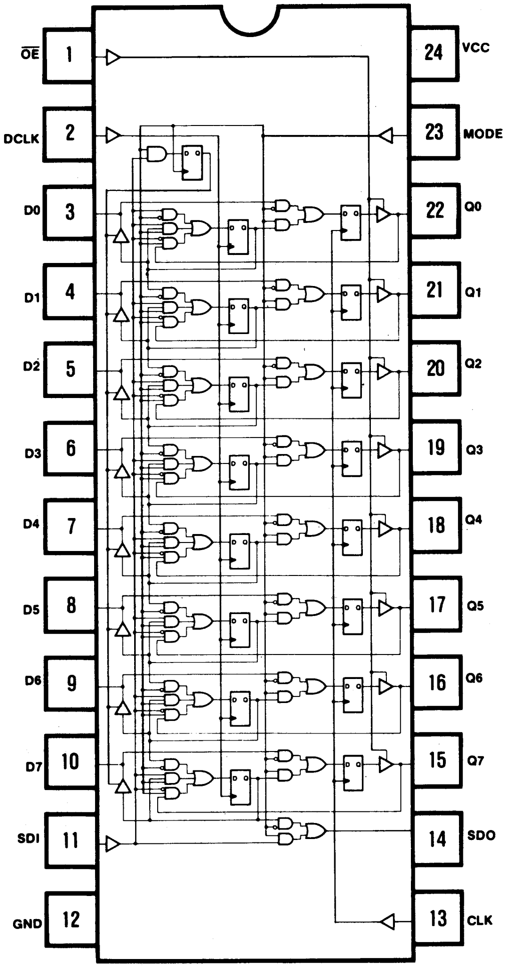


Figure 12c. SN54/74S818 8-Bit Diagnostic Register

Function Table

INPUTS				OUTPUTS			OPERATION
MODE	SDI	CLK	DCLK	Q_3-Q_0	S_3-S_0	SDO	
L	X	↑	*	Qn ← PROM	HOLD	S_3	Load output register from PROM array
L	X	*	↑	HOLD	$S_n \leftarrow S_{n-1}$ $S_0 \leftarrow$ SDI	S_3	Shift shadow register data
L	X	↑	↑	Qn ← PROM	$S_n \leftarrow S_{n-1}$ $S_0 \leftarrow$ SDI	S_3	Load output register from PROM array while shifting shadow register data
H	X	↑	*	$Q_n \leftarrow S_n$	HOLD	SDI	Load output register from shadow register
H	L	*	↑	HOLD	$S_n \leftarrow Q_n$	SDI	Load shadow register from output bus
H	H	*	↑	HOLD	HOLD	SDI	No operation

* Clock must be steady or falling.

Table 1. Diagnostic Components Function Table

Cascadability of the Diagnostic Parts

One very significant feature of the diagnostic parts is their cascadability. Diagnostics is not done very frequently. Therefore, it is very costly to put many data and control lines and ICs on a board just for testing. One way to minimize the cost is by having one input line and one output line and shift in all the bits serially. This means that the SDO of a diagnostic chip must be able to connect to the SDI of another diagnostic chip. Noting that SDI can be both the data input or the control input, SDO must contain the most significant bit of the shadow register if SDI is the data input, and must pass the content of SDI if SDI is used as a control signal.

There is only one data input and one data output to the diagnostic parts. When serial data is shifted in or shifted out, data has to be passed from a diagnostic chip to another. Since SDI must be passed from chip to chip (if it is used for control), it is necessary for logic designers to make sure the fall-through time of SDI to the last chip and the setup time from SDI to DCLK are satisfied.

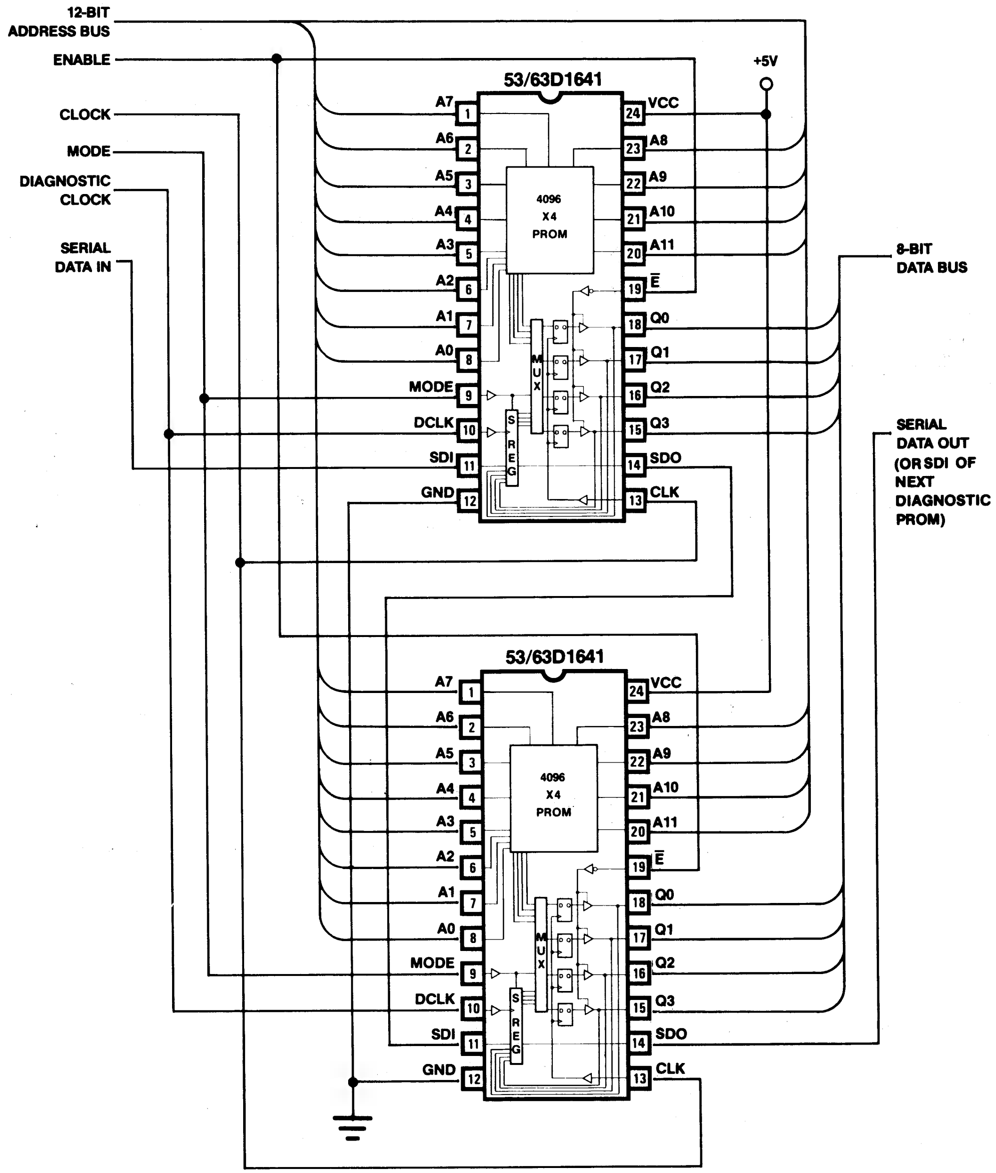


Figure 13. Cascading Diagnostic PROMS for Greater Width

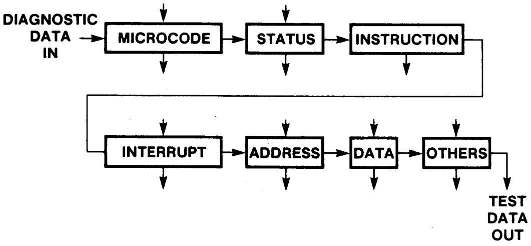
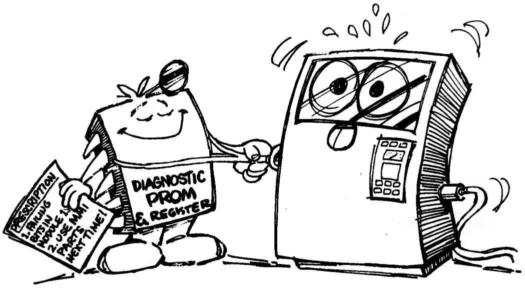


Figure 14. A Way to Link Together Various Diagnostic Parts



"...THE DIAGNOSTIC PROMS AND DIAGNOSTIC REGISTERS HELP YOU TO ANALYZE YOUR SYSTEMS CONVENIENTLY !..."

Some Application Examples

A 4096 word by 64 bits wide microcontroller can be constructed using sixteen 4096 x 4 Diagnostic PROMs (53/63D1641) and one Programmable Array Logic (PAL®) chip. This controller supplies thirty-six control signals, four status select bits, and two addresses of twelve bits each (figure 15) — one for the Next address and one for the Jump address.

In this design, three PROMs are used to store the Next address while an additional three PROMs are used to store the Jump address. Note that three 4-bit wide PROMs provide sufficient inputs to address the full 4096 words of Microprogram memory. One PROM is used to store four status select inputs to the PAL, which is used as a multiplexer for test conditions. A PAL16C1 or PAL20C1 is ideal for this Test Mux since these parts provide many inputs (sixteen and twenty respectively) and complementary outputs (both true and inverted) polarities. The remaining nine PROMs are used to store the 36-bit Microcontrol word. Note that a Microprogram Sequencer is not used in this architecture

The Microcontrol signals control various parts of the CPU and other external blocks such as memory and I/O. For certain microinstructions, some operations may involve a Jump. The 4-bit status select will select a status bit from the test conditions to a pair of complementary outputs which will enable either the Next address or the Jump address. The address enabled will point to the Next microinstruction in the bank of PROMs. If no conditional Jump is needed, both Next and Jump addresses will be the same.

For example, a Jump will be performed if bit 11 of the test conditions is set, the status select bits will be 1011 (which represent 11) and the status to be tested and its complement will appear on the outputs of the PAL. Noting that the output enables of the Diagnostic PROMs are active LOW, the true PAL output controls the Next address PROMs, while the inverted PAL output controls the Jump address PROMs, If the test status is TRUE, the true PAL output disables the Next address PROMs while inverted PAL output enables the Jump address PROMs. The reverse will occur when test status is FALSE. this Next/Jump decision is illustrated in figure 16.

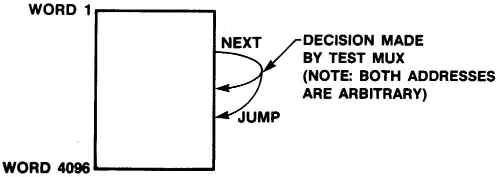

WORD 1
NEXT
JUMP
DECISION MADE BY TEST MUX (NOTE: BOTH ADDRESSES ARE ARBITRARY)
WORD 4096

Figure 16. Microprogram Memory Map

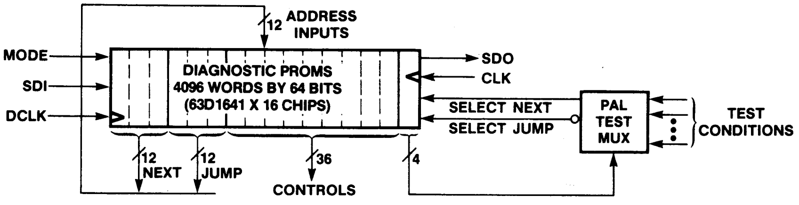


Figure 15. 64-Bit Microcontroller using sixteen 4Kx4 Diagnostic PROMS and One PAL

The decision cycle time is computed by the following equation:

$$f_{MAX} = \frac{1}{t_{su} + t_{CLK} + t_{PD} + t_{PXZ}}$$

where

t_{su} = address setup time for the diagnostic PROM

t_{CLK} = clock to output delay of the PROM

t_{PD} = propagation delay in the outside logic

t_{PXZ} = output enable/disable delay for the diagnostic PROM.

Note that the decision time can be decreased if the Next/Jump decision is made one clock cycle ahead and stored using a synchronous enable. This scheme will reduce the decision time by an amount equal to the propagation delay through the PAL Test Mux, but microcoding this system will become much more complex.

Fewer PROMs would be required if an even/odd Jump address scheme were used. However this decreases the flexibility of PROM addressing.

Another more detailed example of how the diagnostic parts can be built into a system is shown in figure 17a, b, c. The diagnostic parts are used to substitute the instruction register, memory data registers, status register, memory address registers, and writable control store (figure 17a), or registered PROMs for the nonwritable microprogram control store (figure 17b). The only additional block to a typical system without diagnostic features is the diagnostic controller. The diagnostic controller should be able to supply the system with signals like MODE, SDI, DCLK, and the register clock (CLK). In other words, the diagnostic controller in itself is a supercontroller of the processing unit. It should also be noted that all feedback paths, except for those for the register files, are broken.

In normal operation, the diagnostic controller will inactivate the diagnostic feature by setting MODE = LOW and disabling DCLK and have the CLK free running.

When diagnostics is needed, the following sequence is performed:

1. Shift in diagnostic test data bit by bit. In order to perform this operation, CLK is disabled; MODE remains LOW; and SDI contains the bit to be shifted in, and DCLK is enabled. This will continue until a full test vector is shifted into the shadow register.
2. MODE switches to HIGH. Then DCLK is disabled and CLK is enabled. The contents of the shadow register, which is the test vector, will be loaded into the output register.
3. The test result is set up at the inputs of the diagnostic registers and the address lines of the Diagnostic PROMs. MODE switches to LOW again. DCLK is still disabled and CLK is still enabled. The test result will be clocked into the output register.
4. MODE is then switched to HIGH. When DCLK is pulsed, the test result now residing in the output register will be loaded intp the shadow register.
5. With MODE switched to LOW and DCLK enabled and CLK disabled, the test result can be shifted out and analyzed while another test vector is shifted in.

To initialize the RAMs in the writable microprogram control store, first load in a sequence of data and address through another controller. The controller disables the outputs from the microprogram sequencer and feeds in the address through another diagnostic register (figure 17c). There is a switch S_1 which switches the SDI to the registers of the writable control store from some other register (in figure 17c, it is the memory address register) to the diagnostic "control store address" register. The initialization data is shifted into the shadow register by resetting MODE to LOW and enabling DCLK. After all data is shifted into the shadow register, MODE and SDI are set HIGH and then followed by a CLK and a DCLK, and a write to control store. The CLK loads the present control store address to the output registers of the "control store address" register, and the MODE = HIGH and SDI = HIGH at the rising edge of the DCLK will enable the inputs to the diagnostic register to outputs.

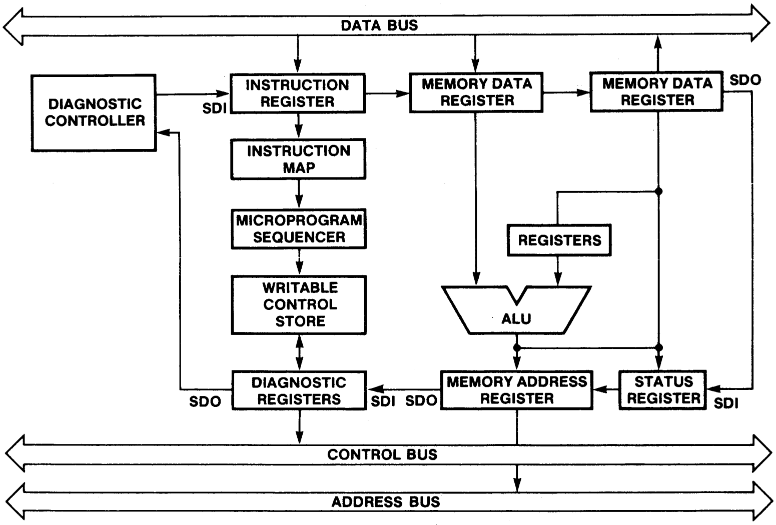


Figure 17a. An Application Example of Using the Diagnostic Registers in a CPU with Writable Control Store

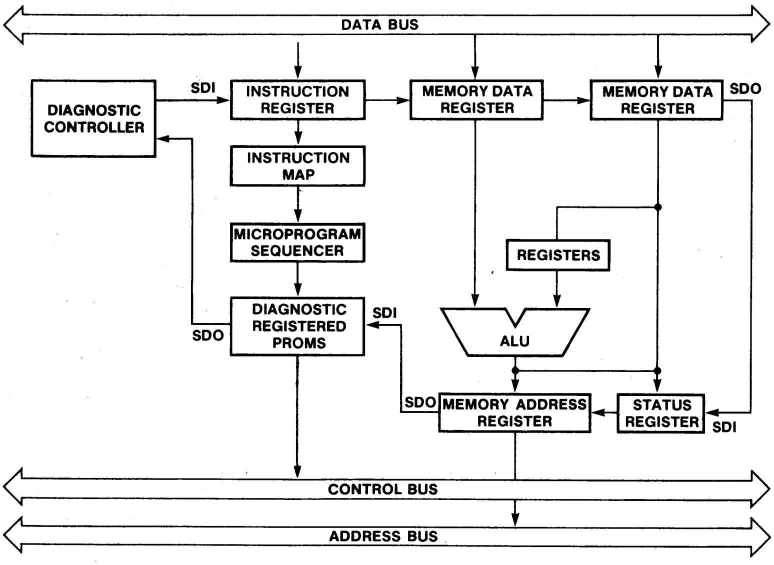


Figure 17b. An Application Example of Using the Diagnostic PROMs and Registers in a CPU

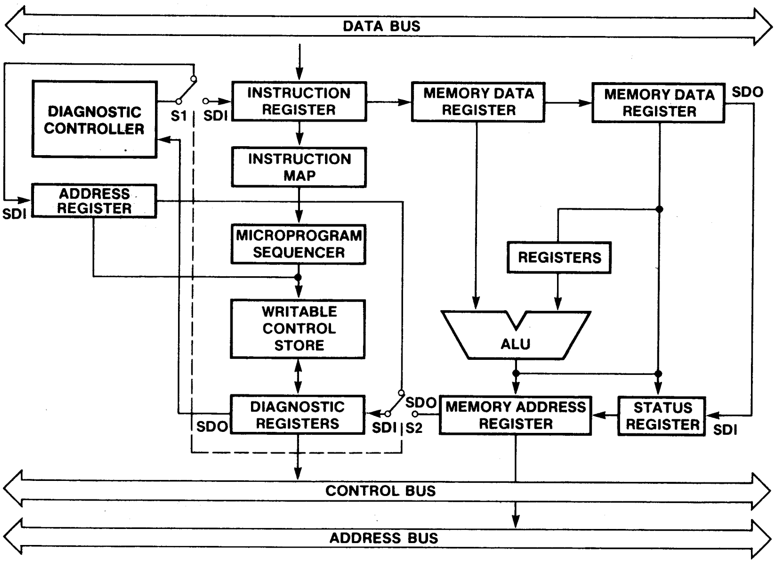


Figure 17c. An Application Example of Using the Diagnostic Registers in a CPU with Writable Control Store. Address Register is Used to Provide the Address for the Writable Initializing Control Store

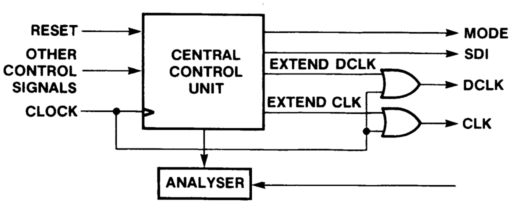


Figure 18. A Block Diagram of the Diagnostics Controller

A block diagram of such a diagnostic controller is shown in figure 18. The central control unit of this controller may be a disk based unit or even another PROM. Noting that in normal operation, MODE remains LOW and only CLK is active, it is possible to include a switch in figure 17a, b, c so that the diagnostic will be inactive even without the diagnostic controller (see figure 19).

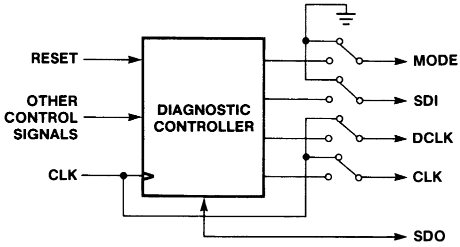


Figure 19. Including a Switch to Disconnect the Diagnostic Controller from the CPU

Some Final Thought

Figure 18 is just a very simple application of the diagnostic parts. A more complicated system may have co-processors, DMA, I/O ports, etc. in addition. The basic theory of using the diagnostic parts is the same — breaking up sequential loops to form a sequential network. In case of a very complicated system, it may be desirable to use a top down approach to solve the diagnostic problem:

1. Break the systems down to subsystems.
2. Use certain test vectors to locate the subsystem which has an error (if exists).
3. Take out the subsystem which contains error.
4. Use some other test vectors to locate the defective part in the subsystem.
5. Replace the defective part and put the subsystem back in.

These parts can be used for diagnostics in most minicomputers, disk drives, terminals, other I/O ports, and even microprocessor based systems.

Besides for diagnostics, the serial shifting feature present in a diagnostic part can also be applied to serial character generator, and other serial to parallel and parallel to serial converter, and serial code generator etc, which will not be discussed in this paper.

Acknowledgements

The authors of this paper would like to give recognition to the circuit designers of the Diagnostic PROM family, namely Bob Bosnyak, Steve Chan, Mike France, and George Geannopoulos.

References

r1. *"Registered PROMs Impact Computer Architecture"* John Birkner, Monolithic Memories Application Note AN-107.

r2. *"Automated Testing of LSI"* R. A. Rasmussen, IEEE Computer Magazine, March 1982, pp. 69-78.

r3. *"Testing Logic Network and Designing for Testability"* Thomas W. Williams and Kenneth P. Parker, IEEE Computer Magazine, October 1979, pp. 9-21.

r4. *"New PROM Architecture Simplies Microprogramming"* John Birkner, Vincent Coli, and Frank Lee, Electro 1983, Session 24.

r5. *"On-chip Circuitry Reveals System's Logic States"* Frank Lee, Vincent Coli, and Warren Miller, Electronic Design, April 14, 1983, pp. 119-124.

r6. *"PAL Programmable Array Logic Handbook"* John Birkner, and Vincent Coli, Monolithic Memories, Inc.

Notes

8-Bit Error Detection and Correction

Bernard Brafman

The explosion of activity in fault-tolerant systems has provoked renewed interest in Error Detection and Correction (EDC) techniques. Of the seversl approaches available, one of the simplest is the Hamming Code. Hamming codes work by introducing redundancy bits in a parallel data word. This application discusses the implementation of such a system using PAL devices.

2175 Mission College Boulevard, Santa Clara, CA 95050 Tel: (408) 970-9700 TWX: 910-338-2376 TWX: 910-338-2374

Monolithic Memories MMI

Single bit error detection and correction for an 8-bit data word requires 4 check bits, making a 12-bit code word. The simplest code to design is a 12-bit Hamming code. To arrive at the code, we set up the following matrix:

	B7	B6	B5	B4	B3	B2	B1	B0	C3	C2	C1	C0
S3	X	X	X	X					X			
S2	X				X	X	X			X		
S1		X	X		X	X		X			X	
S0		X		X	X		X	X				X

The vertical columns are in a counting pattern, excluding the single bit values of 8, 4, 2, 1. The single bit values are assigned to the check bits C3-C0. By reading horizontally across the rows of the matrix, we get the check equations by exclusive OR'ing the data bits with X's in that row and equating that to the check bit with an X in that row:

$$C3 = B7 \oplus B6 \oplus B5 \oplus B4$$
$$C2 = B7 \oplus B3 \oplus B2 \oplus B1$$
$$C1 = B6 \oplus B5 \oplus B3 \oplus B2 \oplus B0$$
$$C0 = B6 \oplus B4 \oplus B3 \oplus B1 \oplus B0$$

The check bits are stored along with the data bits in the following message format:

M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1
B7	B6	B5	B4	C3	B3	B2	B1	C2	B0	C1	C0

When a read occurs, the check bits are recalculated and compared with the stored bits to generate the 4 bit syndrome:

$$S3 = B7 \oplus B6 \oplus B5 \oplus B4 \oplus C3$$
$$S2 = B7 \oplus B3 \oplus B2 \oplus B1 \oplus C2$$
$$S1 = B6 \oplus B5 \oplus B3 \oplus B2 \oplus B0 \oplus C1$$
$$S0 = B6 \oplus B4 \oplus B3 \oplus B1 \oplus B0 \oplus C0$$

The 4 syndrome bits indicate the location of any single bit errors in the 12-bit message format which may than be corrected by inversion.

The Hamming code works by introducing enough other code words to create a difference of exactly 3 bits between legal code words. All other code words are illegal. If, in storage, one bit flips, the result is an illegal word. In addition, there is only one word in the set of legal code words from which it could have come, hence the correction.

The Hamming matrix and resultant sets of check bit and syndrome equations are selected so that when a single bit error occurs, the syndrome gives the position of that bit (either data or check) in the 12-bit message format.

Example	**B7**							**B0**
random data word:	1	1	0	1	1	1	0	0

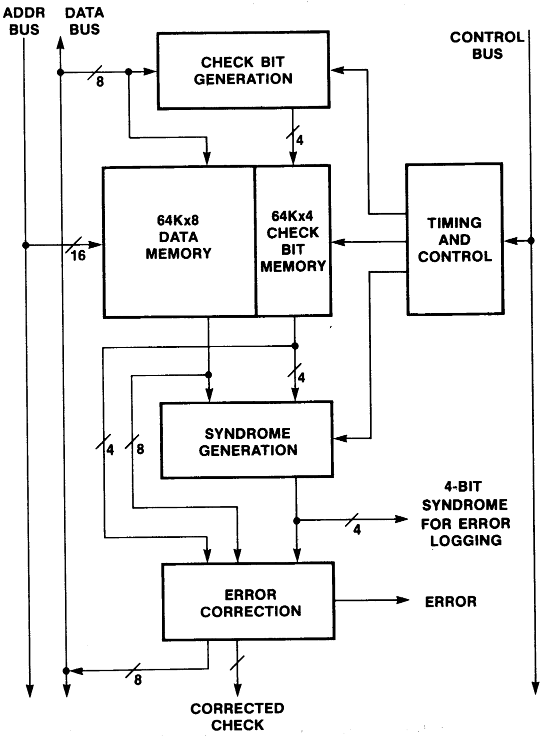
check bits:

$$\begin{aligned} C3 &= B7 \oplus B6 \oplus B5 \oplus B4 \\ &= 1 \oplus 1 \oplus 0 \oplus 1 = 1 \\ C2 &= B7 \oplus B3 \oplus B2 \oplus B1 \\ &= 1 \oplus 1 \oplus 1 \oplus 0 = 1 \\ C1 &= B6 \oplus B5 \oplus B3 \oplus B2 \oplus B0 \\ &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\ C0 &= B6 \oplus B4 \oplus B3 \oplus B1 \oplus B0 \\ &= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 1 \end{aligned}$$

message:

M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1
1	1	0	1	1	1	1	0	1	0	1	1
B7	B6	B5	B4	C3	B3	B2	B1	C2	B0	C1	C0

EDC System Block Diagram



assume no error:

$$\begin{aligned} S3 &= B7 \oplus B6 \oplus B5 \oplus B4 \oplus C3 \\ &= 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0 \\ S2 &= B7 \oplus B3 \oplus B2 \oplus B1 \oplus C2 \\ &= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0 \\ S1 &= B6 \oplus B5 \oplus B3 \oplus B2 \oplus B0 \oplus C1 \\ &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0 \\ S0 &= B6 \oplus B4 \oplus B3 \oplus B1 \oplus B0 \oplus C0 \\ &= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0 \end{aligned}$$

syndrome 0000 indicates no error

assume data bit B7 flips (1 → 0):

$$\begin{aligned} S3 &= \underline{0} \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1 \\ S2 &= \underline{0} \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1 \\ S1 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0 \\ S0 &= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0 \end{aligned}$$

syndrome 1100 indicates M12 or B7 is in error

assume check bit C3 flips (1 → 0):

$$\begin{aligned} S3 &= 1 \oplus 1 \oplus 0 \oplus 1 \oplus \underline{0} = 1 \\ S2 &= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0 \\ S1 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0 \\ S0 &= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0 \end{aligned}$$

syndrome 1000 indicates M8 or C3 is in error

While the check bits can be generated with a 256x4 PROM if the tolerances are loose, high performance systems will need to latch or register the check bits to meet cycle time requirements. Similarly, the syndrome bits can be generated with a 4096x4 PROM but in both cases the PAL16X4 is the high performance choice.

The worst case equations are S1 and S0 with a 6 term exclusive OR. We use 2 properties of the exclusive OR to fit the equations into the 16X4:

1. Associativity
   $A \oplus B \oplus C = (A \oplus B) \oplus C$
2. $A \oplus B \oplus C = A\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}\,\overline{B}C + ABC$

Since a 3 term exclusive OR can be realized with a 4 product sum in sum of products form, a 6 term exclusive OR can be realized by exclusive OR of 2 4 product sums. This is exactly the 16X4 configuration.

Check Bit Equations

$$\begin{aligned} C3 &= B7 \oplus B6 \oplus B5 \oplus B4 \\ &= (B7 \cdot \overline{B6} + \overline{B7} \cdot B6) \oplus (B5 \cdot \overline{B4} + \overline{B5} \cdot B4) \end{aligned}$$

$$\begin{aligned} C2 &= B7 \oplus B3 \oplus B2 \oplus B1 \\ &= (B7 \cdot \overline{B3} + \overline{B7} \cdot B3) \oplus (B2 \cdot \overline{B1} + \overline{B2} \cdot B1) \end{aligned}$$

$$\begin{aligned} C1 &= B6 \oplus B5 \oplus B3 \oplus B2 \oplus B0 \\ &= (B6 \cdot B5 \cdot B3 + \overline{B6} \cdot \overline{B5} \cdot B3 + \overline{B6} \cdot B5 \cdot \overline{B3} + B6 \cdot \overline{B5} \cdot \overline{B3}) \\ &\oplus (B2 \cdot \overline{B0} + \overline{B2} \cdot \quad) \end{aligned}$$

$$\begin{aligned} C0 &= B6 \oplus B4 \oplus B3 \oplus B1 \oplus B0 \\ &= (B6 \cdot B4 \cdot B3 + \overline{B6} \cdot \overline{B4} \cdot B3 + \overline{B6} \cdot B4 \cdot \overline{B3} + B6 \cdot \overline{B4} \cdot \overline{B3}) \\ &\oplus (B1 \cdot \overline{B0} + \overline{B1} \cdot B0) \end{aligned}$$

Syndrome Bit Equations

$$\begin{aligned} S3 &= B7 \oplus B6 \oplus B5 \oplus B4 \oplus C3 \\ &= (B7 \cdot B6 \cdot B5 + \overline{B7} \cdot \overline{B6} \cdot B5 + \overline{B7} \cdot B6 \cdot \overline{B5} + B7 \cdot \overline{B6} \cdot \overline{B5}) \\ &\oplus (B4 \cdot \overline{C3} + \overline{B4} \cdot C3) \end{aligned}$$

$$\begin{aligned} S2 &= B7 \oplus B3 \oplus B2 \oplus B1 \oplus C2 \\ &= (B7 \cdot B3 \cdot B2 + \overline{B7} \cdot \overline{B3} \cdot B2 + \overline{B7} \cdot B3 \cdot \overline{B2} + B7 \cdot \overline{B3} \cdot \overline{B2}) \\ &\oplus (B1 \cdot \overline{C2} + \overline{B1} \cdot C2) \end{aligned}$$

$$\begin{aligned} S1 &= B6 \oplus B5 \oplus B3 \oplus B2 \oplus B0 \oplus C1 \\ &= (B6 \cdot B5 \cdot B3 + \overline{B6} \cdot \overline{B5} \cdot B3 + \overline{B6} \cdot B5 \cdot \overline{B3} + B6 \cdot \overline{B5} \cdot \overline{B3}) \\ &\oplus (\overline{B2} \cdot \overline{B0} \cdot C1 + \overline{B2} \cdot \overline{B0} \cdot C1 + B2 \cdot B0 \cdot \overline{C1} + B2 \cdot \overline{B0} \end{aligned}$$

$$\begin{aligned} S0 &= B6 \oplus B4 \oplus B3 \oplus B1 \oplus B0 \oplus C0 \\ &= (B6 \cdot B4 \cdot B3 + \overline{B6} \cdot \overline{B4} \cdot B3 + \overline{B6} \cdot B4 \cdot \overline{B3} + B6 \cdot \overline{B4} \cdot \overline{B3}) \\ &\oplus (B1 \cdot B0 \cdot C0 + \overline{B1} \cdot \overline{B0} \cdot C0 + \overline{B1} \cdot B0 \cdot \overline{C0} + B1 \cdot \overline{B0} \quad) \end{aligned}$$

The error correction block decodes the 4 syndrome bits, and, if they are not 0000, inverts the indicated bit in the message format. The equations:

$$\begin{aligned} M12\ (= B7C) &= S3 \cdot S2 \cdot \overline{S1} \cdot \overline{S0} \oplus B7 \\ &= S3 \cdot S2 \cdot \overline{S1} \cdot \overline{S0} \cdot \overline{B7} + \overline{(S3 \cdot S2 \cdot \overline{S1} \cdot \overline{S0})} \cdot B7 \\ &= S3 \cdot S2 \cdot \overline{S1} \cdot \overline{S0} \cdot \overline{B7} + \overline{S3} \cdot B7 + S2 \cdot B7 + S1 \\ &\quad \cdot B7 + \overline{S0} \cdot B7 \\ M11\ (= B6C) &= S3 \cdot \overline{S2} \cdot S1 \cdot S0 \cdot \overline{B6} \cdot \overline{S3} \cdot B6 \cdot S2 \cdot B6 + \overline{S1} \\ &\quad \cdot B6 \cdot \overline{S0} \cdot B6 \\ M10\ (= B5C) &= S3 \cdot \overline{S2} \cdot S1 \cdot \overline{S0} \cdot \overline{B5} + \overline{S3} \cdot B5 + S2 \cdot B5 + \overline{S1} \\ &\quad \cdot B5 + S0 \cdot B5 \\ M9\ (= B4C) &= S3 \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot \overline{B4} + \overline{S3} \cdot B4 + S2 \cdot B4 + S1 \\ &\quad \cdot B4 + \overline{S0} \cdot B4 \\ M8\ (= C3C) &= S3 \cdot \overline{S2} \cdot \overline{S1} \cdot \overline{S0} \cdot \overline{C3} + \overline{S3} \cdot C3 + S2 \cdot C3 + S1 \\ &\quad \cdot C3 + S0 \cdot C3 \\ M7\ (= B3C) &= \overline{S3} \cdot S2 \cdot S1 \cdot S0 \cdot \overline{B3} + S3 \cdot B3 + \overline{S2} \cdot B3 + \overline{S1} \\ &\quad \cdot B3 + \overline{S0} \cdot B3 \\ M6\ (= B2C) &= \overline{S3} \cdot S2 \cdot S1 \cdot \overline{S0} \cdot \overline{B2} + S3 \cdot B2 + \overline{S2} \cdot B2 + \overline{S1} \\ &\quad \cdot B2 + S0 + B2 \\ M5\ (= B1C) &= \overline{S3} \cdot S2 \cdot \overline{S1} \cdot S0 \cdot \overline{B1} + S3 \cdot B1 + \overline{S2} \cdot B1 + S1 \\ &\quad \cdot B1 + \overline{S0} \cdot B1 \\ M4\ (= C2C) &= \overline{S3} \cdot S2 \cdot \overline{S1} \cdot \overline{S0} \cdot \overline{C2} + S3 \cdot C2 + \overline{S2} \cdot C2 + S1 \\ &\quad \cdot C2 + S0 \cdot C2 \\ M3\ (= B0C) &= \overline{S3} \cdot \overline{S2} \cdot S1 \cdot S0 \cdot \overline{B0} + S3 \cdot B0 + S2 \cdot B0 + \overline{S1} \\ &\quad \cdot B0 + \overline{S0} \cdot B0 \\ M2\ (= C1C) &= \overline{S3} \cdot \overline{S2} \cdot S1 \cdot \overline{S0} \cdot \overline{C1} + S3 \cdot C1 + S2 \cdot C1 + \overline{S1} \\ &\quad \cdot C1 + S0 \cdot C1 \\ M1\ (= C0C) &= \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot \overline{C0} + S3 \cdot C0 + S2 \cdot C0 + S1 \\ &\quad \cdot C0 \cdot \overline{S0} \cdot C0 \\ ERROR &= \overline{\overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot \overline{S0}} \end{aligned}$$

ERROR is an active high error indicator available for error logging along with the 4 syndrome bits.

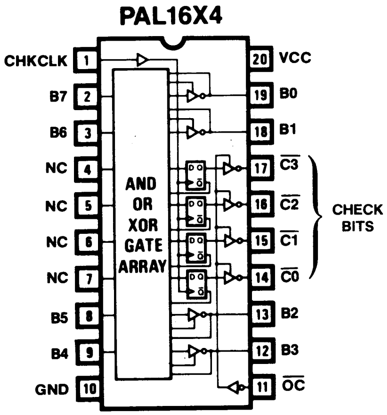
To use 2 PAL16L8's for the error correction block, we need only invert the message bits to get active true outputs.

```
PAL16X4                                              PAL DESIGN SPECIFICATION
CBG                                                     B. BRAFMAN 02/16/81
CHECK BIT GENERATOR
MMI FIELD APPLICATIONS ENGINEER YORBA LINDA, CALIFORNIA
CHKCLK B7 B6 NC NC NC NC B5 B4 GND
/OC B3 B2 /C0 /C1 /C2 /C3 B1 B0 VCC


C3 :=   B7*/B6                ;B7 :+: B6
    +  /B7* B6                ;   :+:
    +   GND                   ;DO NOT BLOW THIS PRODUCT LINE
    +   GND                   ;DO NOT BLOW THIS PRODUCT LINE
   :+:  B5*/B4                ;B5 :+: B4
    +  /B5* B4

C2 :=   B7*/B3                ;B7 :+: B3
    +  /B7* B3                ;   :+:
    +   GND                   ;DO NOT BLOW THIS PRODUCT LINE
    +   GND                   ;DO NOT BLOW THIS PRODUCT LINE
   :+:  B2*/B1                ;B2 :+: B1
    +  /B2* B1

C1 :=   B6* B5* B3            ;B6 :+: B5 :+: B3
    +  /B6*/B5* B3
    +  /B6* B5*/B3            ;   :+:
    +   B6*/B5*/B3
   :+:  B2*/B0                ;B2 :+: B0
    +  /B2* B0

C0 :=   B6* B4* B3            ;B6 :+: B4 :+: B3
    +  /B6*/B4* B3
    +  /B6* B4*/B3            ;   :+:
    +   B6*/B4*/B3
   :+:  B1*/B0                ;B1 :+: B0
    +  /B1* B0
```

```
FUNCTION TABLE

CHKCLK /OC B7 B6 B5 B4 B3 B2 B1 B0 C3 C2 C1 C0

;CONTROL   8 BIT IN   CORR
;CHK  /    BBBBBBBB   CCCC
;CLK  OC   76543210   3210    COMMENTS
-------------------------------------------------
  C   L    HHHHHHHH   LLHH    ALL ONES DATA
  C   L    LHHHHHHH   HHHH    SHIFT A ZERO ACROSS
  C   L    HLHHHHHH   HLLL
  C   L    HHLHHHHH   HLLH
  C   L    HHHLHHHH   HLHL
  C   L    HHHHLHHH   LHLL
  C   L    HHHHHLHH   LHLH
  C   L    HHHHHHLH   LHHL
  C   L    HHHHHHHL   LLLL
  C   L    LLLLLLLL   LLLL    ALL ZEROS DATA
  C   L    HLLLLLLL   HHLL    SHIFT A ONE ACROSS
  C   L    LHLLLLLL   HLHH
  C   L    LLHLLLLL   HLHL
  C   L    LLLHLLLL   HLLH
  C   L    LLLLHLLL   LHHH
  C   L    LLLLLHLL   LHHL
  C   L    LLLLLLHL   LHLH
  C   L    LLLLLLLH   LLHH
-------------------------------------------------
```

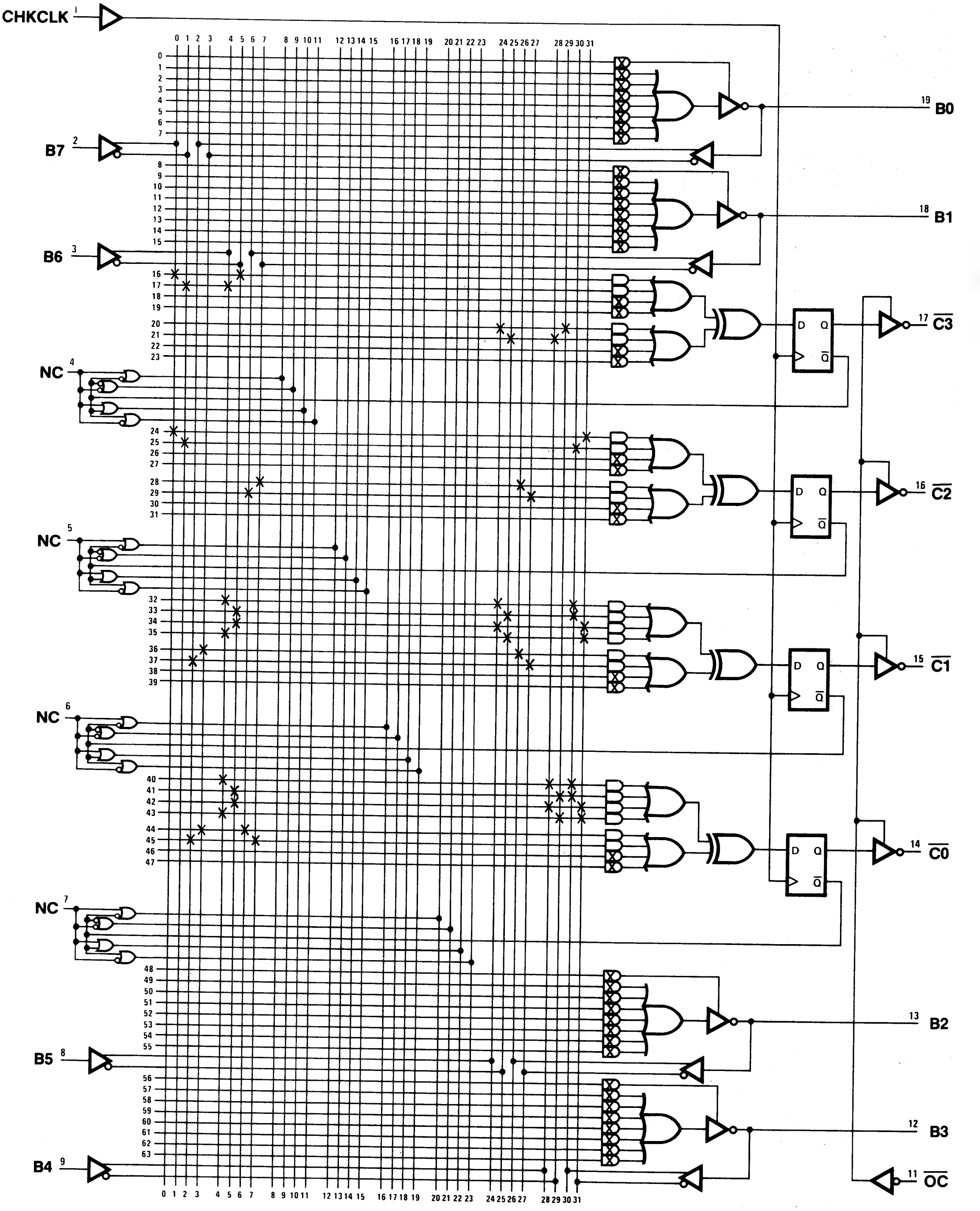
DESCRIPTION

THIS PAL GENERATES THE 4 CHECK BITS IN A 12 BIT HAMMING CODE WORD TO PROVIDE ERROR DETECTION AND CORRECTION ON AN 8 BIT DATA WORD.



Check Bit Generator

Logic Diagram PAL16X4

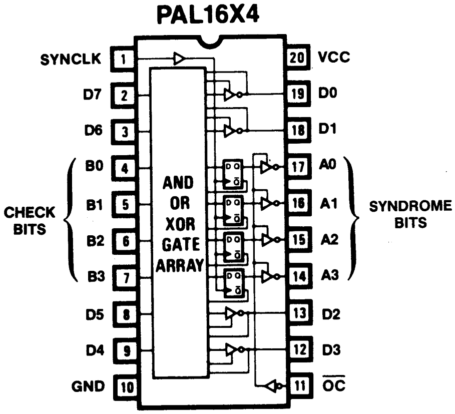


```
PAL16X4                                            PAL DESIGN SPECIFICATION
SBG                                                    B. BRAFMAN 03/13/81
SYNDROME BIT GENERATOR
MMI FIELD APPLICATIONS ENGINEER YORBA LINDA, CALIFORNIA
SYNCLK D7 D6 B0 B1 B2 B3 D5 D4 GND
/OC D3 D2 A3 A2 A1 A0 D1 D0 VCC


;    IN THE ABOVE PIN LIST, THE FOLLOWING SUBSTITUTIONS HAVE BEEN
;    MADE TO ACCOMODATE THE SPECIFIC FORMAT (FIXED SYMBOLS) FOR THE
;    ARITHMETIC PALS IN PALASM:

;         D7 MEANS B7        B0 MEANS /C3   (CHECK BIT 3)
;         D6 MEANS B6        B1 MEANS /C2   (CHECK BIT 2)
;         D5 MEANS B5        B2 MEANS /C1   (CHECK BIT 1)
;         D4 MEANS B4        B3 MEANS /C0   (CHECK BIT 0)
;         D3 MEANS B3        A0 MEANS /S3   (SYNDROME BIT 3)
;         D2 MEANS B2        A1 MEANS /S2   (SYNDROME BIT 2)
;         D1 MEANS B1        A2 MEANS /S1   (SYNDROME BIT 1)
;         D0 MEANS B0        A3 MEANS /S0   (SYNDROME BIT 0)

;  B0-B7 ARE THE BITS OF THE DATA WORD.
;  THE SUBSTITUTIONS APPLY BELOW WITH THE EXCEPTION OF COMMENTS.


/A0 :=    D7* D6* D5               ;B7 :+: B6 :+: B5
     +   /D7*/D6* D5
     +   /D7* D6*/D5               ;    :+:
     +    D7*/D6*/D5
    :+:   D4*(/B0)                 ;B4 :+: C3
     +   /D4*( B0)

/A1 :=    D7* D3* D2               ;B7 :+: B3 :+: B2
     +   /D7*/D3* D2
     +   /D7* D3*/D2               ;    :+:
     +    D7*/D3*/D2
    :+:   D1*(/B1)                 ;B1 :+: C2
     +   /D1*( B1)

/A2 :=    D6* D5* D3               ;B6 :+: B5 :+: B3
     +   /D6*/D5* D3
     +   /D6* D5*/D3               ;    :+:
     +    D6*/D5*/D3
    :+:   D2* D0*( B2)             ;B2 :+: B0 :+: C1
     +   /D2*/D0*( B2)
     +   /D2* D0*(/B2)
     +    D2*/D0*(/B2)

/A3 :=    D6* D4* D3               ;B6 :+: B4 :+: B3
     +   /D6*/D4* D3
     +   /D6* D4*/D3               ;    :+:
     +    D6*/D4*/D3
    :+:   D1* D0*( B3)             ;B1 :+: B0 :+: C0
     +   /D1*/D0*( B3)
     +   /D1* D0*(/B3)
     +    D1*/D0*(/B3)
```

```
FUNCTION TABLE

SYNCLK /OC D7 D6 D5 D4 D3 D2 D1 D0 B0 B1 B2 B3 /A0 /A1 /A2 /A3

;CONTROL   --DATA-          ////
;SYN  /   DDDDDDDD  BBBB   AAAA
;CLK  OC  76543210  0123   0123   COMMENTS
-------------------------------------------
  C   L   HHHHHHHH  LLHH   LLLL   NO ERROR
  C   L   LHHHHHHH  LLHH   HHLL   D7 ERROR
  C   L   HLHHHHHH  LLHH   HLHH   D6 ERROR
  C   L   HHLHHHHH  LLHH   HLHL   D5 ERROR
  C   L   HHHLHHHH  LLHH   HLLH   D4 ERROR
  C   L   HHHHLHHH  LLHH   LHHH   D3 ERROR
  C   L   HHHHHLHH  LLHH   LHHL   D2 ERROR
  C   L   HHHHHHLH  LLHH   LHLH   D1 ERROR
  C   L   HHHHHHHL  LLHH   LLHH   D0 ERROR
  C   L   HHHHHHHH  HLHH   HLLL   B0 ERROR
  C   L   HHHHHHHH  LLHH   LLLL   NO ERROR
  C   L   HHHHHHHH  LLLH   LLHL   B2 ERROR
  C   L   HHHHHHHH  LLHL   LLLH   B1 ERROR
  C   L   LLLLLLLL  LLLL   LLLL   NO ERROR
  C   L   HLLLLLLL  LLLL   HHLL   D7 ERROR
  C   L   LHLLLLLL  LLLL   HLHH   D6 ERROR
  C   L   LLHLLLLL  LLLL   HLHL   D5 ERROR
  C   L   LLLHLLLL  LLLL   HLLH   D4 ERROR
  C   L   LLLLHLLL  LLLL   LHHH   D3 ERROR
  C   L   LLLLLHLL  LLLL   LHHL   D2 ERROR
  C   L   LLLLLLHL  LLLL   LHLH   D1 ERROR
  C   L   LLLLLLLH  LLLL   LLHH   D0 ERROR
  C   L   LLLLLLLL  HLLL   HLLL   B0 ERROR
  C   L   LLLLLLLL  LHLL   LHLL   B1 ERROR
  C   L   LLLLLLLL  LLHL   LLHL   B2 ERROR
  C   L   LLLLLLLL  LLLH   LLLH   B3 ERROR
-------------------------------------------
```

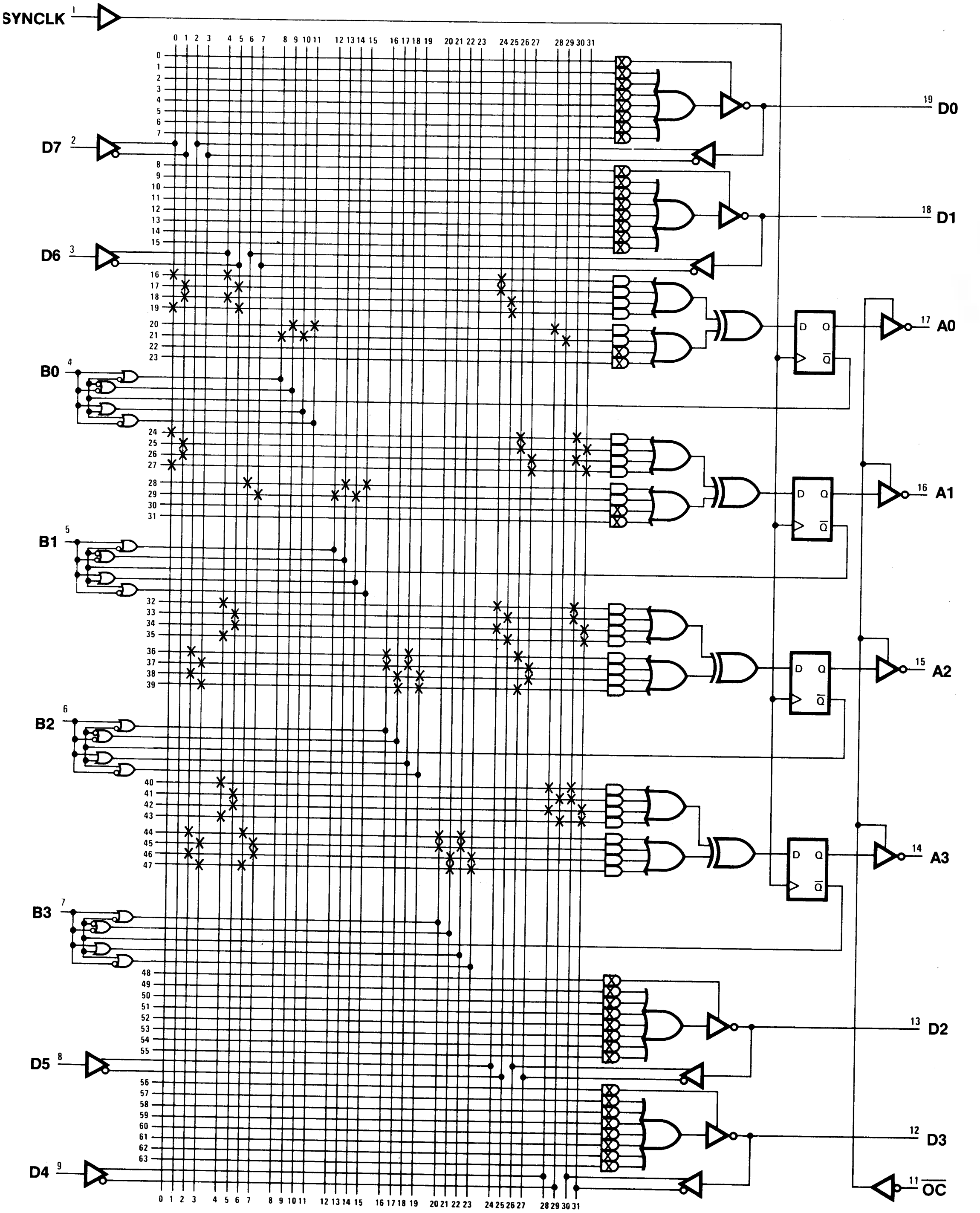
DESCRIPTION

THIS PAL GENERATES THE SYNDROME BITS FOR A 12 BIT HAMMING CODE WORD AS A FUNCTION OF THE 8 DATA BITS AND THE 4 CHECK BITS TO POINT TO ANY SINGLE BIT IN ERROR.



Syndrome Bit Generator

Logic Diagram PAL16X4



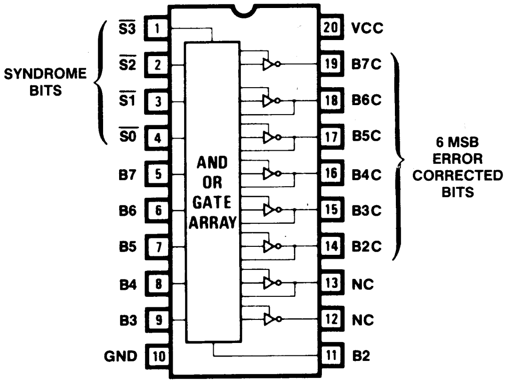
```
PAL16L8                                                 PAL DESIGN SPECIFICATION
ECU1                                                       B. BRAFMAN 03/13/81
ERROR CORRECTION UNIT No. 1
MMI FIELD APPLICATIONS ENGINEER YORBA LINDA, CALIFORNIA
/S3 /S2 /S1 /S0 B7  B6  B5  B4  B3  GND
 B2  NC  NC B2C B3C B4C B5C B6C B7C VCC


IF (VCC) /B7C =  S3* S2*/S1*/S0* B7       ;CORRECTION OF B7
              + /S3*/B7                   ;NO CORRECTION
              + /S2*/B7                   ;NO CORRECTION
              +  S1*/B7                   ;NO CORRECTION
              +  S0*/B7                   ;NO CORRECTION

IF (VCC) /B6C =  S3*/S2* S1* S0* B6       ;CORRECTION OF B6
              + /S3*/B6                   ;NO CORRECTION
              +  S2*/B6                   ;NO CORRECTION
              + /S1*/B6                   ;NO CORRECTION
              + /S0*/B6                   ;NO CORRECTION

IF (VCC) /B5C =  S3*/S2* S1*/S0* B5       ;CORRECTION OF B5
              + /S3*/B5                   ;NO CORRECTION
              +  S2*/B5                   ;NO CORRECTION
              + /S1*/B5                   ;NO CORRECTION
              +  S0*/B5                   ;NO CORRECTION

IF (VCC) /B4C =  S3*/S2*/S1* S0* B4       ;CORRECTION OF B4
              + /S3*/B4                   ;NO CORRECTION
              +  S2*/B4                   ;NO CORRECTION
              +  S1*/B4                   ;NO CORRECTION
              + /S0*/B4                   ;NO CORRECTION

IF (VCC) /B3C = /S3* S2* S1* S0* B3       ;CORRECTION OF B3
              +  S3*/B3                   ;NO CORRECTION
              + /S2*/B3                   ;NO CORRECTION
              + /S1*/B3                   ;NO CORRECTION
              + /S0*/B3                   ;NO CORRECTION

IF (VCC) /B2C = /S3* S2* S1*/S0* B2       ;CORRECTION OF B2
              +  S3*/B2                   ;NO CORRECTION
              + /S2*/B2                   ;NO CORRECTION
              + /S1*/B2                   ;NO CORRECTION
              +  S0*/B2                   ;NO CORRECTION
```

```
FUNCTION TABLE

S3 S2 S1 S0 B7 B6 B5 B4 B3 B2 B7C B6C B5C B4C B3C B2C

;SYNDROME    INPUT     CORRECTED
;  SSSS      BBBBBB     BBBBBB
;  3210      765432     765432        COMMENTS
------------------------------------------------
   LLLL      HHHHHH     HHHHHH        NO ERROR
   LLLL      LLLLLL     LLLLLL        NO ERROR
   HHLL      HHHHHH     LHHHHH        CORRECT B7
   HHLL      LLLLLL     HLLLLL        CORRECT B7
   HLHH      HHHHHH     HLHHHH        CORRECT B6
   HLHH      LLLLLL     LHLLLL        CORRECT B6
   HLHL      HHHHHH     HHLHHH        CORRECT B5
   HLHL      LLLLLL     LLHLLL        CORRECT B5
   HLLH      HHHHHH     HHHLHH        CORRECT B4
   HLLH      LLLLLL     LLLHLL        CORRECT B4
   LHHH      HHHHHH     HHHHLH        CORRECT B3
   LHHH      LLLLLL     LLLLHL        CORRECT B3
   LHHL      HHHHHH     HHHHHL        CORRECT B2
   LHHL      LLLLLL     LLLLLH        CORRECT B2
------------------------------------------------
```

DESCRIPTION

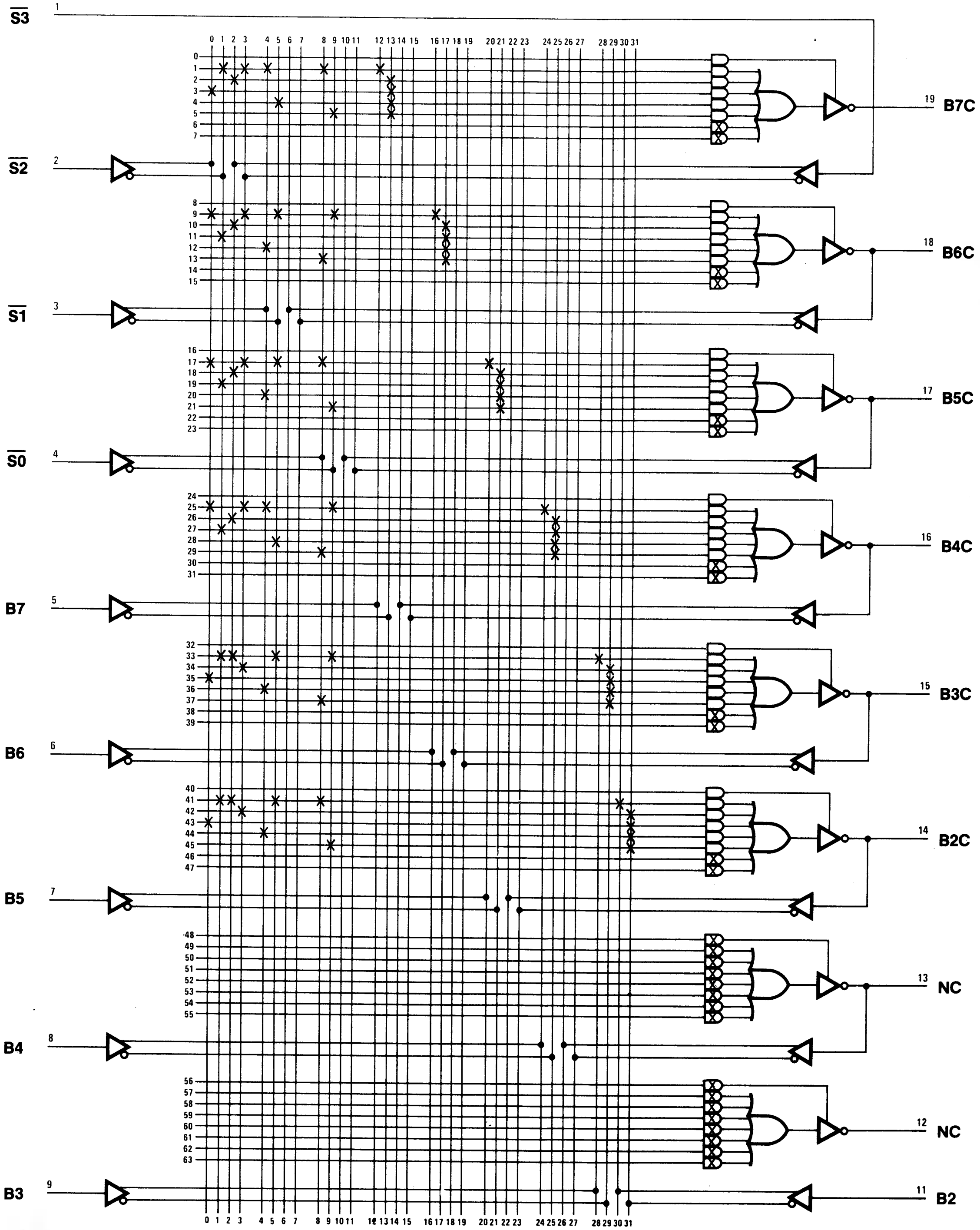
THIS PAL PERFORMS ERROR CORRECTION ON BITS B2-B7 BASED ON THE 4-BIT ERROR SYNDROME S0-S3.

PAL16L8



Error Correction Unit No. 1

Logic Diagram PAL16L8

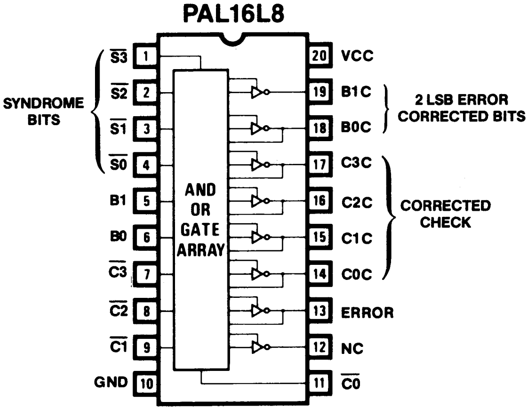


```
PAL16L8                                           PAL DESIGN SPECIFICATION
ECU2                                                  B. BRAFMAN 03/13/81
ERROR CORRECTION UNIT No. 2
MMI FIELD APPLICATIONS ENGINEER  YORBA LINDA, CALIFORNIA
/S3 /S2 /S1 /S0 B1 B0 /C3 /C2 /C1 GND
/C0 NC ERROR C0C C1C C2C C3C B0C B1C VCC


IF (VCC) /B1C = /S3* S2*/S1* S0* B1     ;CORRECTION OF B1
              +  S3*/B1                 ;NO CORRECTION
              + /S2*/B1                 ;NO CORRECTION
              +  S1*/B1                 ;NO CORRECTION

IF (VCC) /B0C = /S3*/S2* S1* S0* B0     ;CORRECTION OF B0
              +  S3*/B0                 ;NO CORRECTION
              +  S2*/B0                 ;NO CORRECTION
              + /S1*/B0                 ;NO CORRECTION
              + /S0*/B0                 ;NO CORRECTION

IF (VCC) /C3C =  S3*/S2*/S1*/S0* C3     ;CORRECTION OF C3
              + /S3*/C3                 ;NO CORRECTION
              +  S2*/C3                 ;NO CORRECTION
              +  S1*/C3                 ;NO CORRECTION
              +  S0*/C3                 ;NO CORRECTION

IF (VCC) /C2C = /S3* S2*/S1*/S0* C2     ;CORRECTION OF C2
              +  S3*/C2                 ;NO CORRECTION
              + /S2*/C2                 ;NO CORRECTION
              +  S1*/C2                 ;NO CORRECTION
              +  S0*/C2                 ;NO CORRECTION

IF (VCC) /C1C = /S3*/S2* S1*/S0* C1     ;CORRECTION OF C1
              +  S3*/C1                 ;NO CORRECTION
              +  S2*/C1                 ;NO CORRECTION
              + /S1*/C1                 ;NO CORRECTION
              +  S0*/C1                 ;NO CORRECTION

IF (VCC) /C0C = /S3*/S2*/S1* S0* C0     ;CORRECTION OF C0
              +  S3*/C0                 ;NO CORRECTION
              +  S2*/C0                 ;NO CORRECTION
              +  S1*/C0                 ;NO CORRECTION
              + /S0*/C0                 ;NO CORRECTION

IF (VCC) /ERROR = /S3*/S2*/S1*/S0       ;NO ERROR!
```

```
FUNCTION TABLE

S3 S2 S1 S0 B1 B0 C3 C2 C1 C0 B1C B0C C3C C2C C1C C0C ERROR

;                        BB    CCCC
;SSSS    BB    CCCC      10    3210
;3210    10    3210      CC    CCCC   ERROR     COMMENTS
---------------------------------------------------------
 LLLL    HH    XXXX      HH    XXXX     L       NO ERROR
 LLLL    XX    HHHH      XX    HHHH     L       NO ERROR
 LLLL    LL    XXXX      LL    XXXX     L       NO ERROR
 LLLL    XX    LLLL      XX    LLLL     L       NO ERROR
 LHLH    HH    XXXX      LH    XXXX     H       CORRECT B1
 LHLH    LL    XXXX      HL    XXXX     H       CORRECT B1
 LLHH    HH    XXXX      HL    XXXX     H       CORRECT B0
 LLHH    LL    XXXX      LH    XXXX     H       CORRECT B0
 HLLL    XX    HHHH      XX    LHHH     H       CORRECT C3
 HLLL    XX    LLLL      XX    HLLL     H       CORRECT C3
 LHLL    XX    HHHH      XX    HLHH     H       CORRECT C2
 LHLL    XX    LLLL      XX    LHLL     H       CORRECT C2
 LLHL    XX    HHHH      XX    HHLH     H       CORRECT C1
 LLHL    XX    LLLL      XX    LLHL     H       CORRECT C1
 LLLH    XX    HHHH      XX    HHHL     H       CORRECT C0
 LLLH    XX    LLLL      XX    LLLH     H       CORRECT C0
---------------------------------------------------------
```

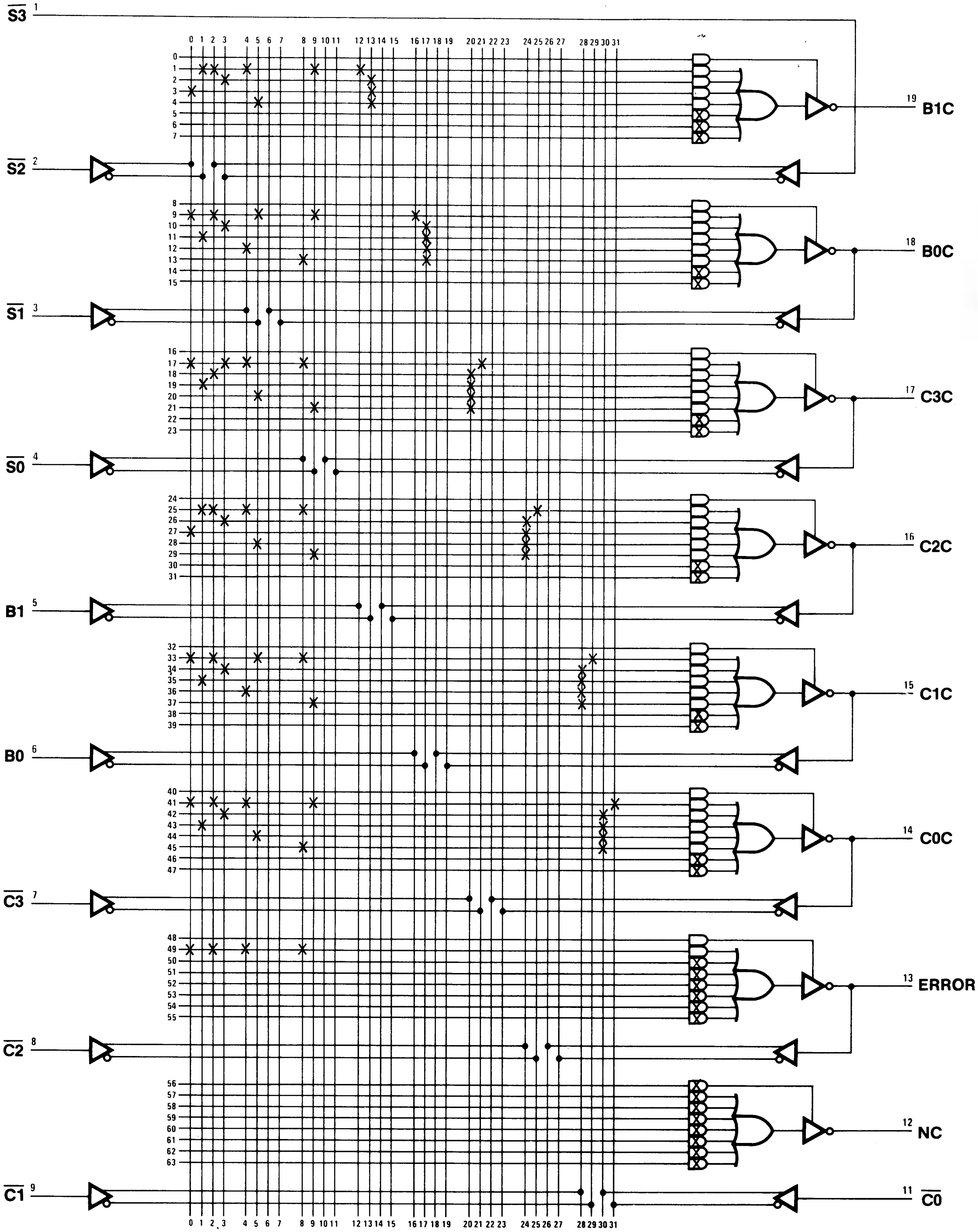
DESCRIPTION

THIS PAL PERFORMS ERROR CORRECTION ON BITS B0-B1 AND CHECKS BITS C0-C3 BASED ON THE 4 BIT ERROR SYNDROME S0-S3.



Error Correction Unit No. 2

Logic Diagram PAL16L8



Notes

32-Bit CRC (Cyclic Redundancy Check)

Nadia Sachs

There is a growing interest in providing data communication links to connect several processors and peripherals into one local area network. One of the most popular networks is the Ethernet. To insure reliable communications in the network an efficient error detection scheme is required. The Ethernet protocol specifies a 32-bit cycle redundancy check (CRC).

The following application opens with a tutorial on the CRC and then shows a detailed implementation of the Ethernet CRC using PAL. The use of fuse programmable devices allows easy modification to accommodate other data communications protocols as well as other applications (CRC in disk drives, etc.).

CYCLIC REDUNDANCY CHECK (CRC) USING PALS

Programmable Array Logic Devices Provide Efficient Implementation of Popular Local Network Error Checking Protocol

There is a growing interest in providing data communication links to connect several processors and peripherals into one local area network. One of the most popular networks is the Ethernet. To insure reliable communications in the network an efficient error detection scheme is required. The Ethernet protocol specifies a 32-bit cycle redundancy check (CRC).

The following application opens with a tutorial on the CRC and then shows a detailed implementation of the Ethernet CRC using PAL. The use of fuse programmable devices allows easy modification to accommodate other data communications protocols as well as other applications (CRC in disk drives, etc.).

Introduction

The growing number of high speed digital links and the need for reliable communication require implementation of efficient error detection schemes. A 32-bit CRC using PAL circuits meets these requirements.

What is CRC ?

CRC is the acronym for Cyclic Redundancy Check, an error detection technique widely used in serial communication systems from computer to computer or from computer to peripheral devices. This technique operates on serial bits of information treated as the coefficients of a binary polynomial, P(x), and processes these bits in modulo-2 arithmetic.

The basic coding concept of the CRC is to modify the polynomial, P(x), so that it is exactly divisible by a fixed polynomial, G(x); the divisor G(x) is referred to as the generator polynomial. The modified polynomial, M(x), is transmitted. M(x) is divided by the same G(x) when received or fetched. If the remainder is zero, all bits are assumed to be correct; otherwise, a flag is set to indicate an error.

An Example Using CRC

A 6-bit message (101110) can be represented in polynomial form as:

$$P(x) = 1 + 0\,x + 1\,x^2 + 1\,x^3 + 1\,x^4 + 0\,x^5$$

or

$$P(x) = x^4 + x^3 + x^2 + 1$$

The highest power of x is attached to the least significant bit (LSB). The LSB is the first bit transmitted in communication channels. In general P(x) will not be exactly divisible by G(x); the division will generate a quotient, Q(x), and a remainder, R(x). P(x) is prescaled to insure that the order of P(x) is greater than the order of G(x) so that the remainder is always different than the message itself.

Specifically, P(x) is prescaled by x^n where n is the degree of G(x).

$$x^n\,P(x) = Q(x)\,G(x) + R(x) \qquad (1)$$

For a 3-bit CRC and $G(x) = x^3 + 1$ equation (1) becomes:

$$x^3\,(x^4 + x^3 + x^2 + 1) = (x^4 + x^3 + x^2 + x)\,(x^3 + 1) + (x^2 + x)$$

The operation is performed using modulo-2 arithmetic where the sum and difference are synonymous.
Equation 1 can equivalently be written as:

$$x^n\,P(x) - R(x) = x^n\,P(x) + R(x) = Q(x)\,G(x) = M(x)$$

M(x) is exactly divisible by G(x) and it is M(x) that is transmitted. The message M(x) is formed by adding the remaining bits, R(x), of a fixed length n to the message bits. Because the message was prescaled, addition is equivalent to appending the remainder at the end of the data bits. For the example given (remembering that LSB is sent first) the information transmitted is: 011 101110. Three redundancy bits are appended to facilitate error detection.

In conclusion when performing CRC, the transmitter will generate and append R(x) while the receiver will verify the exact division of M(x) by G(x) and signal the occurrence of an error.

Why CRC ?

Compared to other error detection schemes such as parity checking CRC is more powerful:

- all errors within n successive bits are detected
- for even G(x) all errors with an odd number of bits in error are detected (50% of all possible random errors)
- all error patterns that are not divisible by G(x) are detected

CRC is also more efficient (for large frame of information). Efficiency is defined as the number of data bits divided by the total number of bits transmitted. For example: in the Ethernet specification the number of data bits ranges from 60 to 1500 bytes, the total number of bits transmitted will contain only an extra 32-bits (if 32-bit CRC is used) instead of an extra 60 to 1500 parity bits (if we assume one parity bit per byte of information is used.)

Implementation

A binary division that ignores the quotient but retains the remainder, R(x), is conventionally implemented with a shift register. A feedback path from the output of the last stage returns to the input of the stages corresponding to the powers of x that have non zero coefficient in the generating polynomial. At each such input, a modulo-2 adder (Exclusive OR) combines the feedback signal with the output from the previous stage.

This is equivalent to a shift subtract on each clock cycle. The example in Fig. 1 shows a 32-bit CRC where G (x), the generator polynominal, (used in ETHERNET and AUTOBAUND II) is given by:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Transmitter operation:

1. Initialize shift register to all Hs. (INIT = H)
2. Data is shifted in the feedback shift register to generate R(x) and fed to the output. (Control C = H)
3. After last data bit has been processed, the complimented R(x) is shifted out for transmission. (Control C = L)

Receiver operation:

1. Initialize shift register to all Hs.
2. Data is shifted in the feedback shift register to regenerate R(x).
3. After last data bit has been processed, Control remains High and the complement of R(x) is shifted (as generated by transmitter).

If the two R(x) match, the final content of the shift register is:

```
X31                              X0
  11000111000001001101110101111011    (residue)
```

4) This value is tested. (Check = L indicates no error, as shown in Fig. 2)

The same hardware can perform both the transmitter and receiver function if the respective inputs are multiplexed.

PAL Configuration

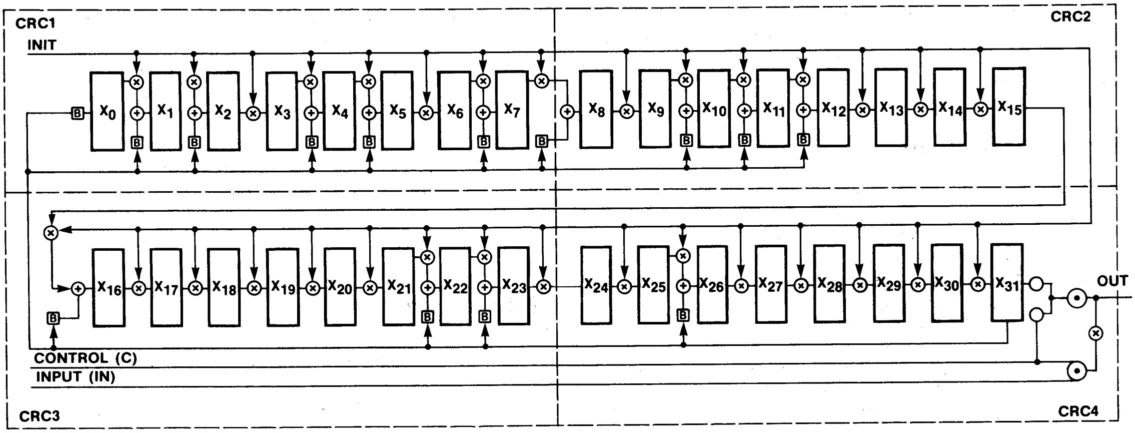
Four PAL circuits are used to implement the 32-bit CRC. The PAL interconnections are shown in Fig. 2. Eight bits of the 32-bit shift register are distributed in each PAL. The logic to check for the residue is distributed among the four PAL circuits as shown in Fig. 3.

Summary

The lack of standardization in data communication equipment makes the use of PAL devices very attractive due to their flexibility and ease of design modification.

REFERENCES

The Ethernet, A Local Area Network, Data Link Layer and Physical Layer Specifications, version 1.0 (joint publication of Digital Equipment Corp., Intel Corp., and Xerox Corp., 1980).



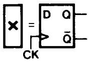
⊗ = OR

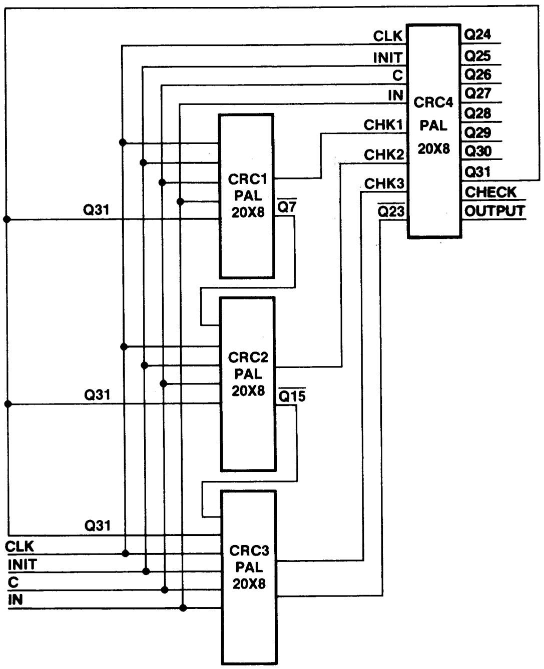
⊕ = XOR

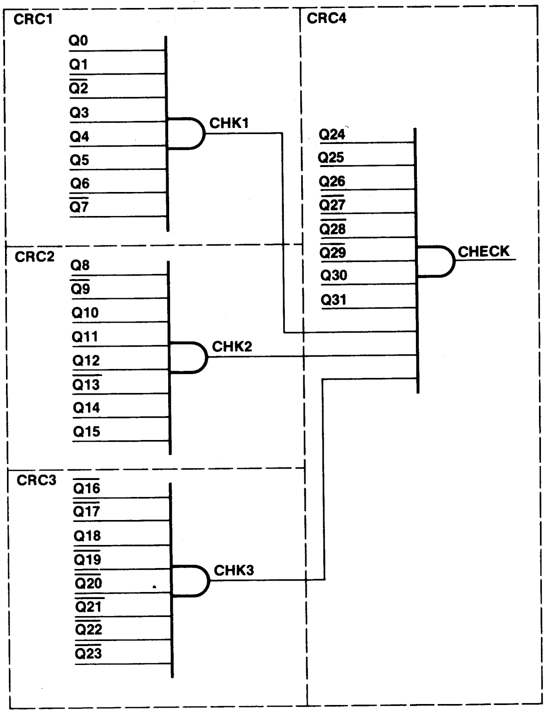
$\boxed{B} = (IN \oplus Q_{31}) \cdot C \cdot \overline{INIT} \equiv IN \cdot Q_{31} \cdot C \cdot \overline{INIT} + \overline{IN} \cdot Q_{31} \cdot C \cdot \overline{INIT}$

⊙ = AND

○ = NOT

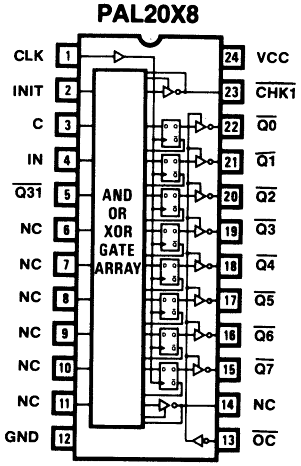






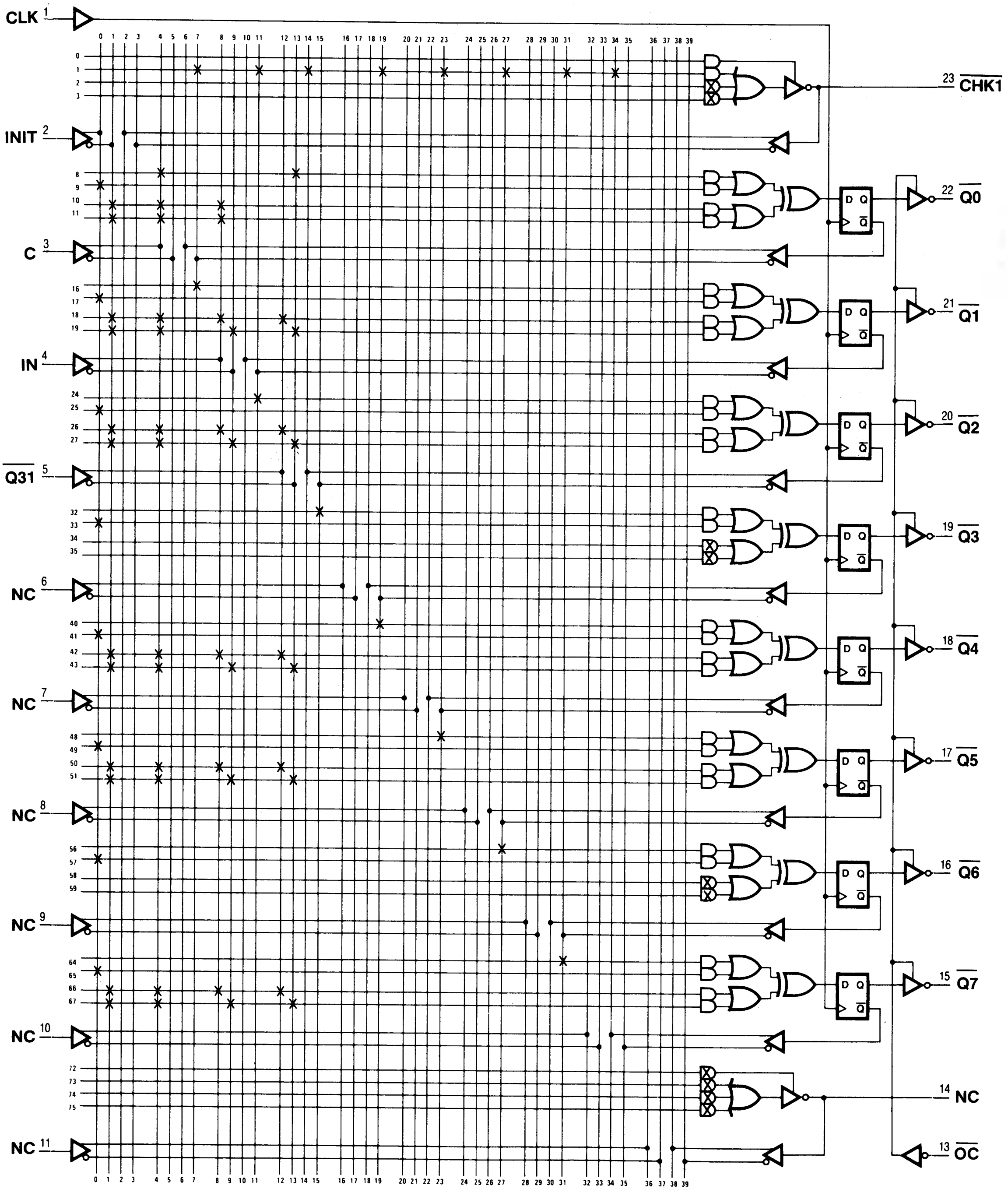
```
PAL20X8                                                PAL DESIGN SPECIFICATION
CRC1                                                       NADIA SACHS 08/14/81
32-BIT CRC (CYCLICAL REDUNDANCY CHECKING) ERROR DETECTION, CHIP 1
MMI SUNNYVALE, CALIFORNIA
CLK INIT C IN /Q31 NC NC NC NC NC NC GND
/OC NC /Q7 /Q6 /Q5 /Q4 /Q3 /Q2 /Q1 /Q0 /CHK1 VCC


IF(VCC) CHK1 =  Q0* Q1*/Q2* Q3* Q4* Q5* Q6*/Q7    ;CHECK BIT 1

Q0  :=   Q31* C                                   ;SHIFT IF C
     +   INIT                                     ;INITIALIZE
    :+: /INIT* IN* C                              ;MODULO-2
     +  /INIT* IN* C                              ;ADDITION

Q1  :=   Q0                                       ;SHIFT
     +   INIT                                     ;INITIALIZE
    :+: /INIT* C* IN*/Q31                         ;MODULO-2
     +  /INIT* C*/IN* Q31                         ;ADDITION

Q2  :=   Q1                                       ;SHIFT
     +   INIT                                     ;INITIALIZE
    :+: /INIT* C* IN*/Q31                         ;MODULO-2
     +  /INIT* C*/IN* Q31                         ;ADDITION

Q3  :=   Q2                                       ;SHIFT
     +   INIT                                     ;INITIALIZE

Q4  :=   Q3                                       ;SHIFT
     +   INIT                                     ;INITIALIZE
    :+: /INIT* C* IN*/Q31                         ;MODULO-2
     +  /INIT* C*/IN* Q31                         ;ADDITION

Q5  :=   Q4                                       ;SHIFT
     +   INIT                                     ;INITIALIZE
    :+: /INIT* C* IN*/Q31                         ;MODULO-2
     +  /INIT* C*/IN* Q31                         ;ADDITION

Q6  :=   Q5                                       ;SHIFT
     +   INIT                                     ;INITIALIZE

Q7  :=   Q6                                       ;SHIFT
     +   INIT                                     ;INITIALIZE
    :+: /INIT* C* IN*/Q31                         ;MODULO-2
     +  /INIT* C*/IN* Q31                         ;ADDITION
```

```
FUNCTION TABLE

CLK /OC INIT C IN Q31 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 CHK1

;                                   Q
;                                   3   QQQQQQQQ
;CLK   /OC    INIT    C    IN      1   76543210    CHK1
-------------------------------------------------------
  C     L      H      X    X       X   HHHHHHHH     X
  C     L      L      H    L       H   LHLLHLLH     X
  C     L      L      H    L       H   LLHLLHLH     X
  C     L      L      H    L       H   HHHHHHLH     X
  C     L      L      H    L       H   LHLLHHLH     X
  C     L      L      H    L       H   LLHLHHLH     X
  C     L      L      H    L       H   HHHLHHLH     X
  C     L      L      H    L       L   HHLHHLHL     X
  C     L      L      H    L       L   HLHHLHLL     X
  C     L      L      H    L       L   LHHLHLLL     X
  C     L      L      H    L       H   LHHLLHHH     X
  C     L      L      H    L       L   HHLLHHHL     X
  C     L      L      H    L       L   HLLHHHLL     X
  C     L      L      H    L       H   HLLLHHHH     X
  C     L      L      H    L       H   HLHLHLLH     X
  C     L      L      H    L       H   HHHLLHLH     X
  C     L      L      H    L       H   LHHHHHLH     X
  C     L      L      H    L       L   HHHHHLHL     X
  C     L      L      H    L       L   HHHHLHLL     X
  C     L      L      H    L       L   HHHLHLLL     X
  C     L      L      H    L       L   HHLHLLLL     X
  C     L      L      H    L       L   HLHLLLLL     X
  C     L      L      H    L       L   LHLLLLLL     X
  C     L      L      H    L       L   HLLLLLLL     X
  C     L      L      H    L       L   LLLLLLLL     X
  C     L      L      H    L       H   HLHHLHHH     X
  C     L      L      H    L       L   LHHLHHHL     X
  C     L      L      H    L       H   LHHLHLHH     X
  C     L      L      H    L       H   LHHLLLLH     X
  C     L      L      H    L       L   HHLLLLHL     X
  C     L      L      H    L       H   LLHHLLHH     X
  C     L      L      H    L       L   LHHLLHHL     X
  C     L      L.     H    L       H   LHHHHLHH     H
-------------------------------------------------------
```



32-Bit CRC, Chip 1

Logic Diagram PAL20X 8



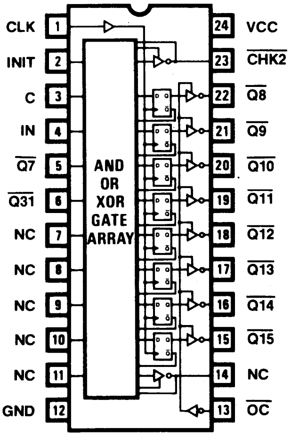
```
PAL20X8                                          PAL DESIGN SPECIFICATION
CRC2                                                NADIA SACHS 08/14/81
32-BIT CRC (CYCLICAL REDUNDANCY CHECKING) ERROR DETECTION, CHIP 2
MMI SUNNYVALE, CALIFORNIA
CLK INIT C IN /Q7 /Q31 NC NC NC NC NC GND
/OC NC /Q15 /Q14 /Q13 /Q12 /Q11 /Q10 /Q9 /Q8 /CHK2 VCC


IF(VCC) CHK2 =  Q8*/Q9* Q10* Q11* Q12*/Q13* Q14* Q15   ;CHECK BIT 2

Q8  :=   Q7                                            ;SHIFT
    +    INIT                                          ;INITIALIZE
   :+: /INIT* C* IN*/Q31                               ;MODULO-2
    +  /INIT* C*/IN* Q31                               ;ADDITION

Q9  :=   Q8                                            ;SHIFT
    +    INIT                                          ;INITIALIZE

Q10 :=   Q9                                            ;SHIFT
    +    INIT                                          ;INITIALIZE
   :+: /INIT* C* IN*/Q31                               ;MODULO-2
    +  /INIT* C*/IN* Q31                               ;ADDITION

Q11 :=   Q10                                           ;SHIFT
    +    INIT                                          ;INITIALIZE
   :+: /INIT* C* IN*/Q31                               ;MODULO-2
    +  /INIT* C*/IN* Q31                               ;ADDITION

Q12  :=  Q11                                           ;SHIFT
    +    INIT                                          ;INITIALIZE
   :+: /INIT* C* IN*/Q31                               ;MODULO-2
    +  /INIT* C*/IN* Q31                               ;ADDITION

Q13 :=   Q12                                           ;SHIFT
    +    INIT                                          ;INITIALIZE

Q14 :=   Q13                                           ;SHIFT
    +    INIT                                          ;INITIALIZE

Q15 :=   Q14                                           ;SHIFT
    +    INIT                                          ;INITIALIZE
```

```
FUNCTION TABLE

CLK /OC INIT C IN Q7 Q31 Q15 Q14 Q13 Q12 Q11 Q10 Q9 Q8 CHK2

;                                  Q   QQQQQQQQ
;                              Q   3   111111
;CLK   /OC    INIT   C    IN   7   1   54321098   CHK2
-----------------------------------------------------
  C     L      H      X    X    X   X   HHHHHHHH    X
  C     L      L      H    L    H   H   HHHLLLHL    L
  C     L      L      H    L    L   H   HHLHHLLH    L
  C     L      L      H    L    L   H   HLHLHHHH    X
  C     L      L      H    L    H   H   LHLLLLHL    X
  C     L      L      H    L    L   H   HLLHHLLH    X
  C     L      L      H    L    L   H   LLHLHHHH    X
  C     L      L      H    L    H   L   LHLHHHHH    X
  C     L      L      H    L    H   L   HLHHHHHH    X
  C     L      L      H    L    H   L   LHHHHHHH    X
  C     L      L      H    L    L   H   HHHLLLHH    X
  C     L      L      H    L    L   L   HHLLLHHL    X
  C     L      L      H    L    H   L   HLLLHHLH    X
  C     L      L      H    L    H   H   LLLLLHHL    X
  C     L      L      H    L    H   H   LLLHLLLL    X
  C     L      L      H    L    H   H   LLHHHHLL    X
  C     L      L      H    L    H   H   LHHLLHLL    X
  C     L      L      H    L    L   L   HHLLHLLL    X
  C     L      L      H    L    H   L   HLLHLLLH    X
  C     L      L      H    L    H   L   LLHLLLHH    X
  C     L      L      H    L    H   L   LHLLLHHH    X
  C     L      L      H    L    H   L   HLLLHHHH    X
  C     L      L      H    L    H   L   LLLHHHHH    X
  C     L      L      H    L    L   L   LLHHHHHL    X
  C     L      L      H    L    H   L   LHHHHHLH    X
  C     L      L      H    L    L   H   HHHLLHHH    X
  C     L      L      H    L    H   L   HHLLHHHH    X
  C     L      L      H    L    L   H   HLLLLLHH    X
  C     L      L      H    L    L   H   LLLHHLHH    X
  C     L      L      H    L    L   L   LLHHLHHL    X
  C     L      L      H    L    H   H   LHHHLLLL    X
  C     L      L      H    L    L   L   HHHLLLLL    X
  C     L      L      H    L    L   H   HHLHHHLH    H
-----------------------------------------------------
```

DESCRIPTION

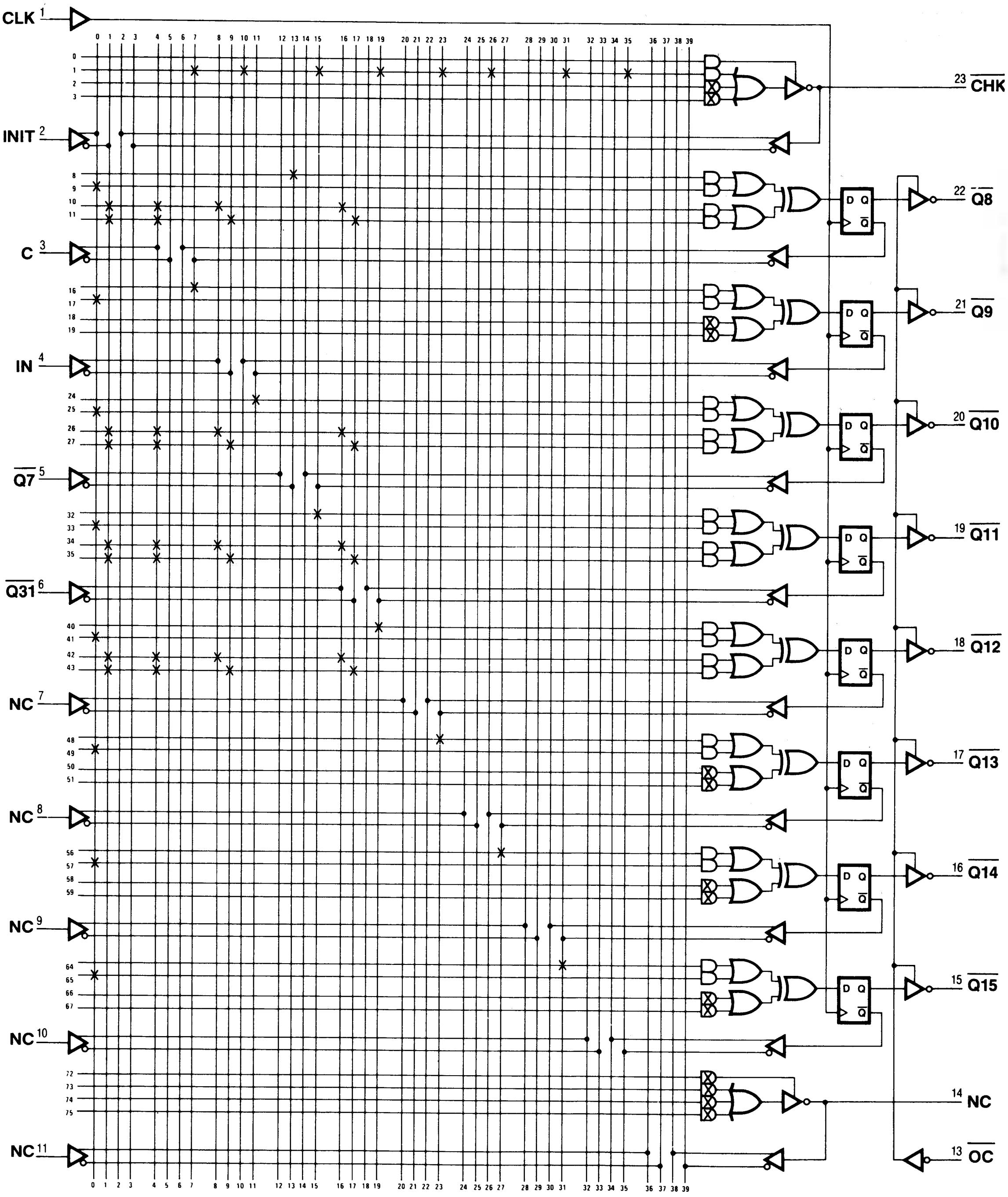
SECOND 8-BIT SHIFT REGISTER AND CHECK.

PAL20X8



32-Bit CRC, Chip 2

Logic Diagram PAL20X 8



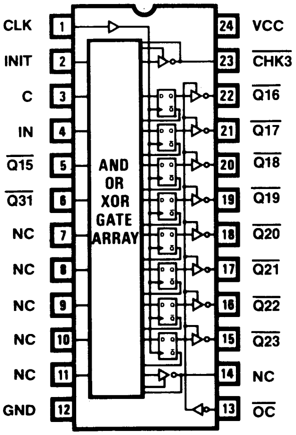
```
PAL20X8                                            PAL DESIGN SPECIFICATION
CRC3                                                   NADIA SACHS 08/14/81
32-BIT CRC (CYCLICAL REDUNDANCY CHECKING) ERROR DETECTION CHIP 3
MMI SUNNYVALE,CALIFORNIA
CLK INIT C IN /Q15 /Q31 NC NC NC NC NC GND
/OC NC /Q23 /Q22 /Q21 /Q20 /Q19 /Q18 /Q17 /Q16 /CHK3 VCC


IF(VCC) CHK3  = /Q16*/Q17* Q18*/Q19*/Q20*/Q21*/Q22*/Q23   ;CHECK BIT 3

Q16  :=   Q15                                             ;SHIFT
      +   INIT                                            ;INITIALIZE
     :+: /INIT* C* IN*/Q31                                ;MODULO-2
      +  /INIT* C*/IN* Q31                                ;ADDITION

Q17  :=   Q16                                             ;SHIFT
      +   INIT                                            ;INITIALIZE

Q18  :=   Q17                                             ;SHIFT
      +   INIT                                            ;INITIALIZE

Q19  :=   Q18                                             ;SHIFT
      +   INIT                                            ;INITIALIZE

Q20  :=   Q19                                             ;SHIFT
      +   INIT                                            ;INITIALIZE

Q21  :=   Q20                                             ;SHIFT
      +   INIT                                            ;INITIALIZE

Q22  :=   Q21                                             ;SHIFT
      +   INIT                                            ;INITIALIZE
     :+: /INIT* C* IN*/Q31                                ;MODULO-2
      +  /INIT* C*/IN* Q31                                ;ADDITION

Q23  :=   Q22                                             ;SHIFT
      +   INIT                                            ;INITIALIZE
     :+: /INIT* C* IN*/Q31                                ;MODULO-2
      +  /INIT* C*/IN* Q31                                ;ADDITION
```

```
FUNCTION TABLE

CLK /OC INIT C IN Q15 Q31 Q23 Q22 Q21 Q20 Q19 Q18 Q17 Q16 CHK3

;                                  Q    Q    QQQQQQQQ
;                                  1    3    22221111
;CLK    /OC    INIT    C    IN     5    1    32109876    CHK3
-----------------------------------------------------------------
  C      L      H      X    X      X    X    HHHHHHHH     X
  C      L      L      H    L      H    H    LLHHHHHL     L
  C      L      L      H    L      H    H    HLHHHHLL     L
  C      L      L      H    L      H    H    HLHHHLLL     X
  C      L      L      H    L      H    H    HLHHLLLL     X
  C      L      L      H    L      L    H    HLHLLLLH     X
  C      L      L      H    L      H    H    HLLLLLHL     X
  C      L      L      H    L      L    L    LLLLLHLL     X
  C      L      L      H    L      L    L    LLLLHLLL     X
  C      L      L      H    L      H    L    LLLHLLLH     X
  C      L      L      H    L      L    H    HHHLLLHH     X
  C      L      L      H    L      H    L    HHLLLHHH     X
  C      L      L      H    L      H    L    HLLLHHHH     X
  C      L      L      H    L      H    H    HHLHHHHL     X
  C      L      L      H    L      L    H    LHHHHHLH     X
  C      L      L      H    L      L    H    LLHHHLHH     X
  C      L      L      H    L      L    H    HLHHLHHH     X
  C      L      L      H    L      L    L    LHHLHHHL     X
  C      L      L      H    L      H    L    HHLHHHLH     X
  C      L      L      H    L      H    L    HLHHHLHH     X
  C      L      L      H    L      L    L    LHHHLHHL     X
  C      L      L      H    L      L    L    HHHLHHLL     X
  C      L      L      H    L      H    L    HHLHHLLH     X
  C      L      L      H    L      L    L    HLHHLLHL     X
  C      L      L      H    L      L    L    LHHLLHLL     X
  C      L      L      H    L      L    H    LLLLHLLH     X
  C      L      L      H    L      H    L    LLLHLLHH     X
  C      L      L      H    L      H    H    HHHLLHHL     X
  C      L      L      H    L      H    H    LLLLHHLL     X
  C      L      L      H    L      L    L    LLLHHLLL     X
  C      L      L      H    L      L    H    HHHHLLLH     X
  C      L      L      H    L      L    L    HHHLLLHL     X
  C      L      L      H    L      H    H    LLLLLHLL     H
-----------------------------------------------------------------
```

DESCRIPTION

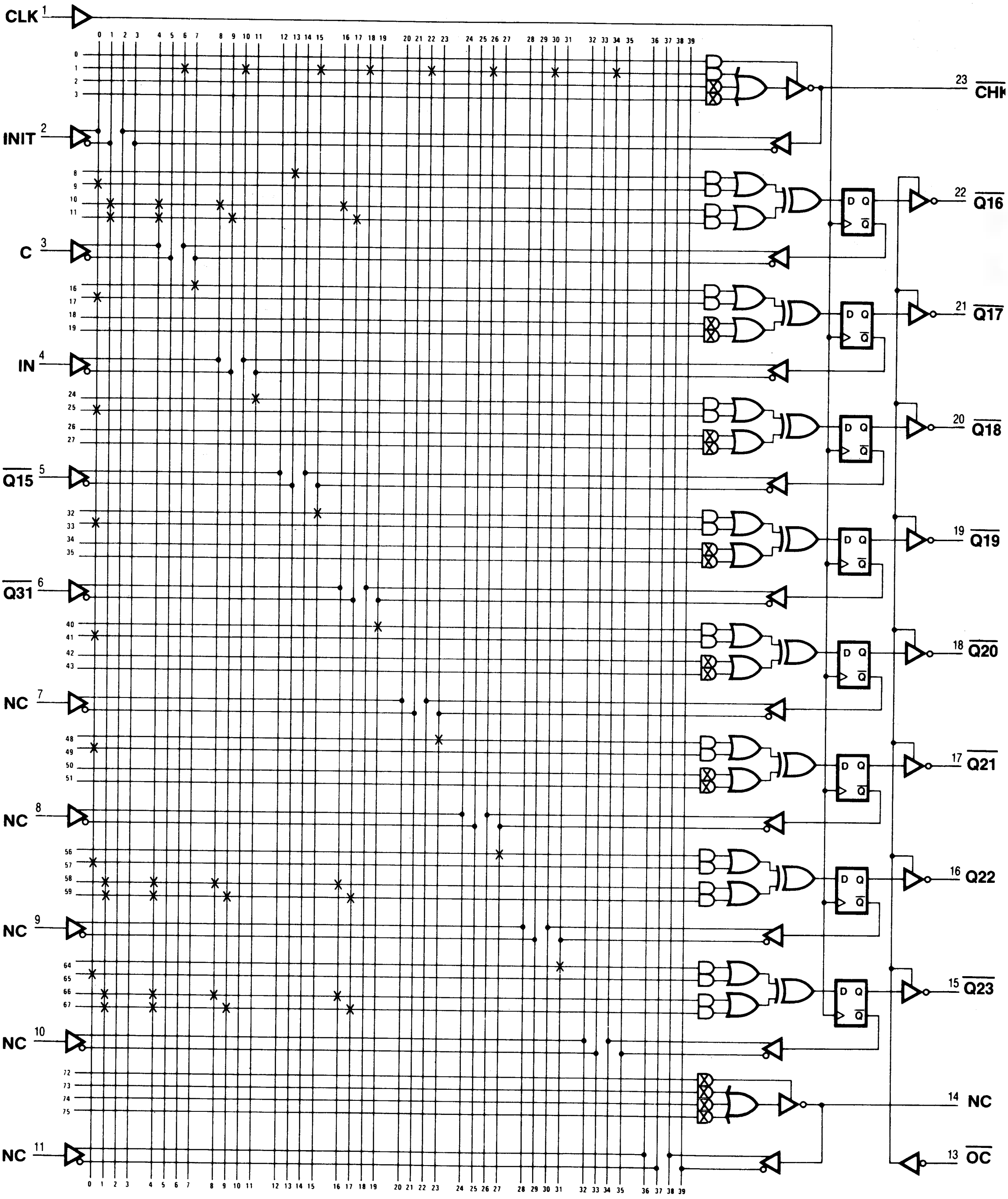
THIRD 8-BIT SHIFT REGISTER AND CHECK.

PAL20X8



32-Bit CRC, Chip 3

Logic Diagram PAL20X8

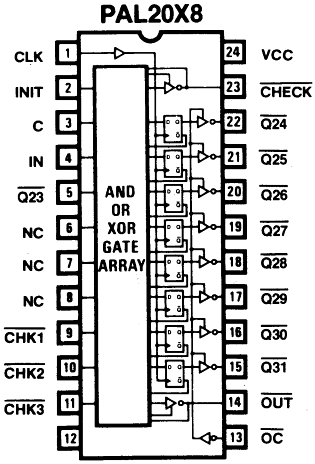


```
PAL20X8                                                   PAL DESIGN SPECIFICATION
CRC4                                                         NADIA SACHS 08/14/81
32-BIT CRC (CYCLICAL REDUNDANCY CHECKING) ERROR DETECTION, CHIP 4
MMI SUNNYVALE, CALIFORNIA
CLK INIT C IN /Q23 NC NC NC /CHK1 /CHK2 /CHK3 GND
/OC /OUT /Q31 /Q30 /Q29 /Q28 /Q27 /Q26 /Q25 /Q24 /CHECK VCC


IF(VCC) CHECK  =   CHK1* CHK2* CHK3* Q24* Q25* Q26*/Q27     ;CHECK
               *                 /Q28*/Q29* Q30* Q31     ;ERROR

Q24    :=   Q23                                             ;SHIFT
        +   INIT                                            ;INITIALIZE

Q25    :=   Q24                                             ;SHIFT
        +   INIT                                            ;INITIALIZE

Q26    :=   Q25                                             ;SHIFT
        +   INIT                                            ;INITIALIZE
       :+: /INIT* C* IN*/Q31                                ;MODULO-2
        +  /INIT* C*/IN* Q31                                ;ADDITION

Q27    :=   Q26                                             ;SHIFT
        +   INIT                                            ;INITIALIZE

Q28    :=   Q27                                             ;SHIFT
        +   INIT                                            ;INITIALIZE

Q29    :=   Q28                                             ;SHIFT
        +   INIT                                            ;INITIALIZE

Q30    :=   Q29                                             ;SHIFT
        +   INIT                                            ;INITIALIZE

Q31    :=   Q30                                             ;SHIFT
        +   INIT                                            ;INITIALIZE

IF(VCC) OUT =  Q31*/C                                       ;SERIAL
           + /IN * C                                        ;OUT
```

```
FUNCTION TABLE

CLK /OC INIT C IN CHK1 CHK2 CHK3 Q23 Q31 Q30 Q29 Q28 Q27 Q26 Q25 Q24 OUT CHECK

;                                           Q    QQQQQQQQ
;                                    CHK    2    33222222
;CLK    /OC    INIT    C     IN      123    3    10987654    OUT    CHECK
--------------------------------------------------------------------------
  C      L      H      X     L       XXX    X    HHHHHHHH     X       X
  C      L      L      H     L       XXX    H    HHHHHLHH     H       X
  C      L      L      H     L       XXX    L    HHHHLLHL     H       X
  C      L      L      H     L       XXX    H    HHHLLLLH     H       X
  C      L      L      H     L       XXX    H    HHLLLHHH     H       X
  C      L      L      H     L       XXX    H    HLLLHLHH     H       X
  C      L      L      H     L       XXX    H    LLLHLLHH     H       X
  C      L      L      H     L       XXX    H    LLHLLHHH     H       X
  C      L      L      H     L       XXX    L    LHLLHHHL     H       X
  C      L      L      H     L       XXX    L    HLLHHHLL     H       X
  C      L      L      H     L       XXX    L    LLHHHHLL     H       X
  C      L      L      H     L       XXX    H    LHHHHLLH     H       X
  C      L      L      H     L       XXX    H    HHHHLLHH     H       X
  C      L      L      H     L       XXX    H    HHHLLLHH     H       X
  C      L      L      H     L       XXX    H    HHLLLLHH     H       X
  C      L      L      H     L       XXX    L    HLLLLLHL     H       X
  C      L      L      H     L       XXX    L    LLLLLLLL     H       X
  C      L      L      H     L       XXX    H    LLLLLLLH     H       X
  C      L      L      H     L       XXX    L    LLLLLLHL     H       X
  C      L      L      H     L       XXX    H    LLLLLHLH     H       X
  C      L      L      H     L       XXX    H    LLLLHLHH     H       X
  C      L      L      H     L       XXX    L    LLLHLHHL     H       X
  C      L      L      H     L       XXX    H    LLHLHHLH     H       X
  C      L      L      H     L       XXX    H    LHLHHLHH     H       X
  C      L      L      H     L       XXX    H    HLHHLHHH     H       X
  C      L      L      H     L       XXX    L    LHHLHLHL     H       X
  C      L      L      H     L       XXX    L    HHLHLHLL     H       X
  C      L      L      H     L       XXX    L    HLHLHHLL     H       X
  C      L      L      H     L       XXX    H    LHLHHHLH     H       X
  C      L      L      H     L       XXX    L    HLHHHLHL     H       X
  C      L      L      H     L       XXX    L    LHHHLLLL     H       X
  C      L      L      H     L       XXX    H    HHHLLLLH     H       X
  C      L      L      H     L       HHH    H    HHLLLHHH     H       H
  C      L      L      L     X       XXX    X    HLLLHHHX     H       X
  C      L      L      L     X       XXX    X    LLLHHHXX     L       X
--------------------------------------------------------------------------
```

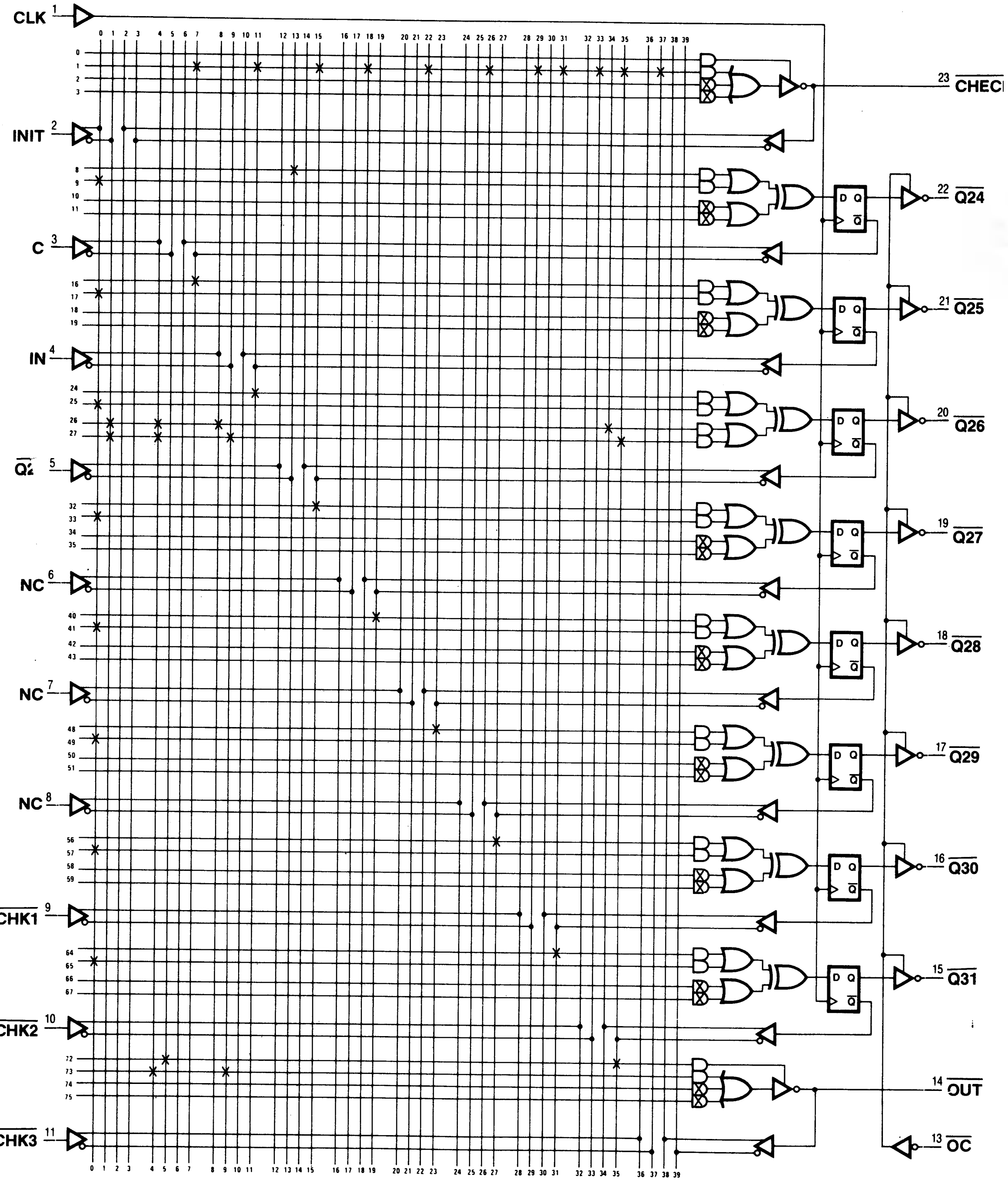
DESCRIPTION

FOURTH 8-BIT SHIFT REGISTER, CHECK, AND SERIAL OUT.



32-Bit CRC, Chip 4

Logic Diagram PAL20X8



Implementation of Serial/Parallel CRC Using PAL Devices

Vivian Kong

Abstract

CRC, or Cycling Redundancy Check, is an error detection technique widely used in digital data communication and storage systems. CRC can be performed either serially or in parallel. Serial CRC is implemented in an environment where data is transmitted in a bit-wise manner. In systems where data is transmitted in form of bytes, it is more desirable to implement CRC in parallel. The following article will describe the hardware required for implementing both serial and parallel CRC. It will then discuss how the family of Programmable Array Logic devices can be applied in such implementations. Detailed PAL design examples of a serial CRC-16 generator and an 8-bit parallel CRC-CCITT generator are included in the appendix for reference.

Introduction

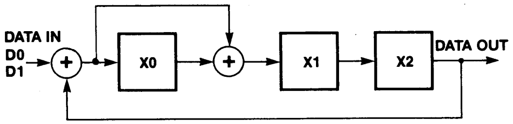
In digital transmission and storage systems, errors can be introduced in the transmitted or stored data by different sources of noise, or, by the hardware itself. Owing to the growing demand for reliability in these systems, great emphasis has been put on the different error control methods to ensure that data has been accurately transmitted or stored. Some of these control methods will detect certain types of errors in the transmitted data, while other methods will correct the erroneus data as well. CRC, or Cycling Redundancy Check, belongs to the former group of error control methods.

What is CRC?

CRC is a commonly used error detection technique because it is fairly easy to implement and it has a high error detection capability. The CRC scheme can be described briefly as follows. A predetermined number, say, N, of CRC bits are first generated from the data to be transmitted. These N redundancy check bits are then transmitted together with the data. On data reception, the same number of CRC bits are regenerated from the transmitted data. It is then compared against the previously generated CRC bits. Errors occur when these two CRC bit patterns are not the same. If errors are detected, usually a retransmission will be attempted.

CRC is applied in data communication and storage systems. These storage systems include floppy and other disk storage systems, digital cassette and cartridge tape systems. In serial communication, it is natural to perform CRC in serial. In systems, such as cartridge tape systems, where data is transmitted in bytes, it is faster and more efficient to perform CRC in a byte-wise, parallel fashion. Let us first look at how the CRC bits are generated by hardware serially in the 3-bit CRC example shown in Figure 1.0.

How to generate CRC bits serially?



INIT. COND.	X0	X1	X2
1ST SHIFT	$D0 \oplus X2$	$D0 \oplus X2 \oplus X0$	X1
2ND SHIFT	$D1 \oplus X1$	$D1 \oplus X1 \oplus D0 \oplus X2$	$D0 \oplus X2 \oplus X0$

3 REDUNDANCY CHECK BITS

Figure 1.0. An Example Showing How to Generate 3 Redundancy Check Bits Using a 3-Bit Linear Feedback Shift Register (LFSR)

The redundancy check bits can be generated by using a linear feedback shift register or LFSR. Consider the 3-bit LFSR in Figure 1.0. This LFSR is made up of three registers, X0, X1 and X2, and it will generate three CRC bits. The output of the last register stage X2, is XORed with the data bit before feeding back to the input of the first register stage, X0. It is also XORed with the output of register X0, before feeding back to the input of register X1. Assuming that the initial conditions of the registers are X0, X1 and X2. Imagine two data bits, D0 and D1, are shifted into the LFSR bit by bit. The internal state of the LFSR after every shift is shown in figure 1.0. After all data bits are shifted into the LFSR, the bits residing in the registers are the redundancy check bits. In this case, the redundancy check bits are the registers' contents after the second shift, $D1 \oplus X1$, $D1 \oplus X1 \oplus D0 \oplus X2$, $D0 + X2 + X0$. Notice that each CRC bit is a "running" parity check bit that checks on the incoming data bits continuously. Moreover, the parity check patterns generated by these redundancy check bits are fixed by the feedback terms in the LFSR. These CRC bits can then be shifted out from the LFSR by disabling the feedback paths, as described in Appendix B. It is useful to realize that the CRC scheme works with the N-bit LFSR initialized to any random pattern. Furthermore, the redundancy check bits can either be transmitted directly or selectively inverted. In practice, a combination of these methods can be applied.

The hardware of this particular 3-bit LFSR can be represented with the polynomial

$$G(X) = X^0 + X^1 + X^3$$

This polynomial is also known as the generator polynomial. The highest power term in the polynomial indicates the number of registers required in the LFSR, whereas the other power terms represent the positions of the feedback terms. In this case, the number of registers required is three, specified by the X^3 term; the positions of the feedback terms are in registers X0 and X1, specified by the X^0 and X^1 terms. In this way the structure of any N-bit LFSR is transparent by just specifying the generator polynomial, G(X). Therefore, in order to generate N redundancy check bits or N-bit CRC, it is only required to specify the generator polynomial, G(X). A list of the popular generator polynomials can be found in Table 1.

STANDARD	GENERATOR POLYNOMIAL, G(X)
CRC-CCITT	$X^{16} + X^{12} + X^5 + 1$
CRC-CCITT reverse	$X^{16} + X^{11} + X^4 + 1$
CRC-16	$X^{16} + X^{15} + X^{12} + 1$
CRC-16 reverse	$X^{16} + X^{14} + X + 1$
	$X^{16} + X^{15} + X^{13} + X^7 + X^4 + X^2 + X + 1$
CRC-12	$X^{12} + X^{11} + X^3 + X^2 + X + 1$
LRC-8	$X^8 + 1$
	$X^8 + X^7 + X^5 + X^4 + X + 1$

Table 1 Table Showing Popular Generator Polynomial.

These generator polynomials are selected to detect the most probable errors in the system where the CRC is applied. The CRC-CCITT generator polynomial, for example, is commonly used in cartridge tape systems. It can detect all single burst errors within 16 bits. It can also detect all occurrences of an odd number of bits in error. In addition, it can detect all single, double, and triple-bit errors if the record length is less than

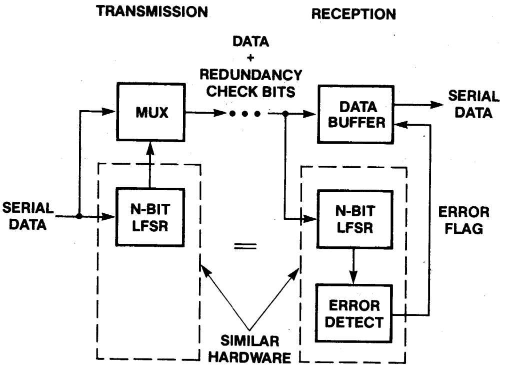


Figure 2.0. Serial CRC

32,767 bits. Finally, it detects over 99.99% of all possible burst length longer than sixteen.

Figure 2.0 shows how serial CRC works. At the same time the serial data is transmitted, it is also shifted into the N-bit LFSR to generate the N redundancy check bits. After all the data is transmitted and shifted into the N-bit LFSR, the redundancy check bits are then shifted out serially from the LFSR for transmission. The received information, which consists of the data and the N redundancy check bits, is again shifted into a similar N-bit LFSR to check for error. The comparison between the CRC bits generated before transmission and during reception is performed automatically within the LFSR during data reception. If there is no error during transmission, the contents residing inside the registers, after all the transmitted data has been processed, should match an expected pattern; otherwise, error occurs.

Serial CRC Hardware

The same hardware can be used to generate the redundancy check bits for transmission and for error detection before reception. This hardware consist of (i) an N-bit LFSR, and (ii) an error detection circuitry which generates an error flag when checking data validity.

Parallel CRC

Let us first give the definition of an M-bit parallel implementation of an N-bit CRC. N is the number of CRC bits, fixed by the generator polynomial, G(X). M is the number of data bits desired to be processed in parallel. M is usually the length of a byte, for example, M can be 8 or 16. M is independent of N, and it is important not to confuse one with another. Figure 3.0 shows a 2-bit parallel implementation of the 3-bit LFSR in Figure 1.0. As described above, after two shifts, the contents of the registers X0, X1, X2 in the 3-bit LFSR are D1 ⊕ X1, D1 ⊕ X1 ⊕ D0 ⊕ X2, and D0 ⊕ X2 ⊕ X0.

The circuitry in Figure 3.0 consists of a 2-bit look-ahead circuit which actually implements the contents of the three registers after two shifts. In fact, any M-bit parallel implementation of a serial LFSR can be made by constructing the M-bit look-ahead circuitry. This M-bit lookahead circuitry can be found by physically shifting M data bits through the serial LFSR or by matrix calculations. Details of the matrix calculations of an 8-bit parallel CRC using the CRC-CCITT standard generator polynomial can be found in Appendix A.

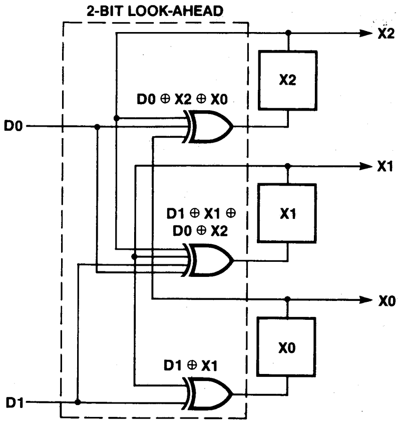


Figure 3.0. Diagram Showing 2-Bit Parallel Implementation of the 3-Bit LFSR in Figure 1.0

Parallel CRC works almost the same as serial CRC, as shown in Figure 4.0. The only difference is that, in parallel CRC data is processed M bits in parallel. Therefore, in general, parallel CRC is faster than serial CRC, depending on the size of M and the complexity of the M-bit look-ahead circuitry.

Parallel CRC Hardware

As in the case of serial CRC, the same hardware can be used to generate the redundancy check bits for transmission and to detect error before reception. The hardware required for implementing an M-bit parallel CRC consists of (i) an M-bit look-ahead circuit, which consists of just XOR gates and N registers; and (ii) an error detection circuit that generates an error flag when checking for errors in the data.

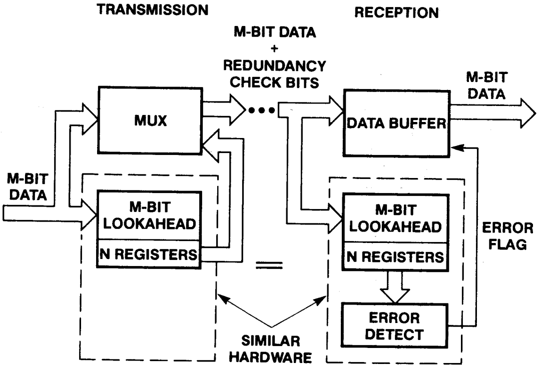


Figure 4.0. Parallel CRC

Why PAL Devices in CRC?

The family of Programming Array Logic devices provides the ease and flexibility in hardware implementations. At present, the standard CRC chips available in the market are only serial CRC with selected G(X). PAL devices give a systematic approach for implementing serial CRC with any generator polynomial, G(X), or any N-bit LFSR. In the case of parallel CRC, the flexible structure in the PAL devices make them good candidates for such implementations.

How to apply PAL circuits in Serial CRC?

The N-bit LFSR composed only of 3-input XOR gates and N registers. The most suitable PAL devices for implementing such logics are the PAL20x family (see the PAL Handbook). The error detection circuitry is merely an N-input AND gate, where the N inputs are from the outputs of the N registers, and the output of the AND gate is optionally latched to form the required error flag. It is assumed that the error flag is latched throughout the examples used in this application note.

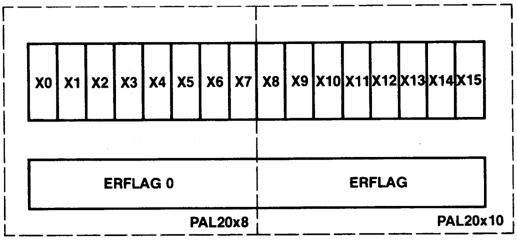


Figure 5.0. Using 2-PAL Devices for Implementing a Serial 16-Bit CRC

To implement a 16-bit CRC requires altogether seventeen registers. Sixteen registers will be for the 16-bit LFSR, whereas the extra register is for the error flag. Eight of the sixteen registers can be put into one PAL device, the others in another PAL device. Since the sixteen registers are put into two PAL devices, the error flag then, must also be implemented in 2 stages. The first for ANDing together the first eight register outputs, the second for ANDing together the second eight register outputs and the output of the first error flag stage. This partitioning is shown in Figure 5.0. The first eight registers (X0, ..., X7) with their corresponding feedback terms (not shown), and the first unlatched error flag stage, ERFLAG0, can be put into a PAL20x8. The second eight registers (X8, ..., X15) and the final latched error flag, ERFLAG, can be all put into a PAL20x10.

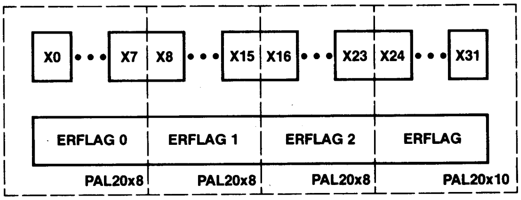


Figure 5.1. Using 4 PAL Devices for Implementing a Serial 32-Bit CRC

Figure 5.1 shows the partitioning for a 32-bit CRC using 4 PAL devices. The logic behind this partitioning is similar to the case of the 16-bit CRC. To give an idea of the number of PAL devices required for implementing an N-bit CRC, the number of PAL devices needed for implementing an 8-bit CRC up to a 48-bit CRC is summarized in Table 2. Detailed PAL design of a 16-bit CRC using the CRC-16 standard polynomial is given in Appendix B. The PAL design for the 32-bit CRC using the Ethernet specifications can be found in the PAL Handbook.

How about PAL devices in Parallel CRC?

Unlike the serial CRC, there is no standard way of arranging a M-bit parallel CRC circuit in PAL devices. This is because of the differences in the complexity of the hardware for parallel CRC with different generator polynomial, G(X). The bulk of the hardware is in the M-bit look-ahead circuitry. Although the M-bit look-ahead circuitry is just XOR gates, the complexity of their arangement depends on the degree and the number of feedback terms in the generator polynomial, G(X), and the size of M. In most cases, the generator polynomial is fixed. The generator polynomial is selected to detect the best probable errors in the system where it is applied. The decision,

CRC OF VARIOUS BITS	NO. OF PAL DEVICES REQUIRED	NO. OF PAL20x8	NO. OF PAL20x10
8 - bits	1	0	1
16 - bits	2	1	1
24 - bits	3	2	1
32 - bits	4	3	1
40 - bits	5	4	1
48 - bits	6	5	1

Table 2. Table Showing the Number of PAL Devices Required to Implement a Serial N-Bit CRC.

then, depends on M. In the case of interfacing a parallel CRC circuitry with a 16-bit data bus, it is almost certainly faster to process data, eight bits, or sixteen bits, in parallel than to process data serially. The hardware required for implementing a 16-bit parallel CRC will be, in most cases, double that required for implementing an 8-bit parallel CRC. The trade-off, then, will be made between speed and cost.

A design example of using four PALs to implement an 8-bit parallel CRC using the CRC-CCITT standard generator polynomial, is given in Appendix C. This generator polynomial, when implemented 8-bit in parallel, gives a relatively "nice" structure, which can be partitioned into four PAL devices. Figure 8.5 in Appendix C shows the interconnections of these PAL devices. The uneven distribution of the input and output pins shows the complexity of this "nice" circuitry, while at the same time shows the flexibility of the PAL devices in handling such complexity. It is worthwhile to mention that for cases with larger M and N, registered-PROMs will be another good alternative for such implementations.

Use PAL devices in CRC!

For Serial CRC, the PAL20x family can implement any generator polynomial easily. For Parallel CRC, the flexible structure of the PAL devices makes it attractive for such implementations, as can be seen in the parallel CRC design example. Therefore, the Programmable Array Logic family is a convenient tool for CRC implementations.

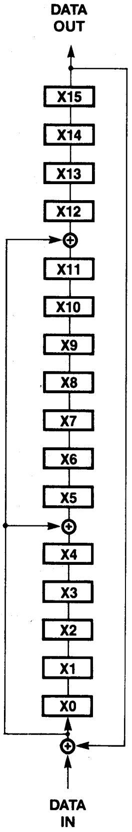


Figure 6.0. A 16-Bit Linear Feedback Shift Register Implementing the CRC-CCITT Standard Generator Polynomial $G(X) = X^{16} + X^{12} + X^5 + 1$

Appendix A

Derivation of the equations for the 8-bit parallel CRC using the CRC-CCITT standard

The generator polynomial for the CRC-CCITT standard is

$$G(X) = X^{16} + X^{12} + X^5 + 1$$

The LFSR required to implement such polynomial in serial CRC is shown in Figure 6.0. There are altogether sixteen registers with the feedback terms at registers X12, X5 and X0.

It is desired, in the 8-bit parallel CRC design example, to process data, eight bits at a time. Therefore, it is necessary to find the equations for implementing the 8-bit look-ahead circuitry. This can be found by shifting eight bits of data, D0, ..., D7, into the LFSR. After eight shifts, the contents of the LFSR can be represented by the following set of equations:

$$\begin{aligned}
X0\ (n+1) &= X8\ (n) \oplus X12(n) \oplus D(3) \oplus D(7)\\
X1\ (n+1) &= X9\ (n) \oplus X13(n) \oplus D(2) \oplus D(6)\\
X2\ (n+1) &= X10(n) \oplus X14(n) \oplus D(1) \oplus D(5)\\
X3\ (n+1) &= X11(n) \oplus X15(n) \oplus D(1) \oplus D(4)\\
X4\ (n+1) &= X12(n) \oplus D(3)\\
X5\ (n+1) &= X8\ (n) \oplus X12(n) \oplus X13(n) \oplus D(2) \oplus D(3) \oplus D(7)\\
X6\ (n+1) &= X9\ (n) \oplus X13(n) \oplus X14(n) \oplus D(1) \oplus D(2) \oplus D(6)\\
X7\ (n+1) &= X10(n) \oplus X14(n) \oplus X15(n) \oplus D(0) \oplus D(1) \oplus D(5)\\
X8\ (n+1) &= X0\ (n) \oplus X11(n) \oplus X15(n) \oplus D(0) \oplus D(4)\\
X9\ (n+1) &= X1\ (n) \oplus X12(n) \oplus D(3)\\
X10(n+1) &= X2\ (n) \oplus X13(n) \oplus D(2)\\
X11(n+1) &= X3\ (n) \oplus X14(n) \oplus D(1)\\
X12(n+1) &= X4\ (n) \oplus X8\ (n) \oplus X12(n) \oplus X15(n) \oplus D(0) \oplus D(3)\\
X13(n+1) &= X5\ (n) \oplus X9\ (n) \oplus X13(n) \oplus D(2) \oplus D(6)\\
X14(n+1) &= X6\ (n) \oplus X10(n) \oplus X14(n) \oplus D(1) \oplus D(5)\\
X15(n+1) &= X7\ (n) \oplus X11(n) \oplus X15(n) \oplus D(0) \oplus D(4)
\end{aligned}$$

________________ equation (0)

where Xi (n + 1) is the next value of the corresponding register i, i = 0, ..., 15
Xi (n) is the present value of the corresponding register i, i = 0, ..., 15
D (n) is the parallel input data bits, where n = 0, ..., 7

These results can also be derived mathematically without actually performing the tedious shifting. The structure of the 16-bit LFSR can be represented in the following matrix form:

This matrix representation of the 16-bit LFSR in figure 6.0 is intuitive. The individual equation for each register, X0, ..., X15, is written and put in a matrix form, equation (1). Each equation can be broken up into two parts, a shift, and a feedback term composed of X15 (n) ⊕ D (n) equation (2). The identity matrix, I, in equation (2), performs a shift, where on each shift, each register stage gets the value of the preceeding register stage. For example, X (n + 1) = X (n). In the case of register X0, there

$$\left|\begin{array}{c} X\,0\,(n+1)\\ X\,1\,(n+1)\\ X\,2\,(n+1)\\ X\,3\,(n+1)\\ X\,4\,(n+1)\\ X\,5\,(n+1)\\ X\,4\,(n+1)\\ \cdot\\ \cdot\\ X11\,(n+1)\\ X12\,(n+1)\\ X13\,(n+1)\\ \cdot\\ \cdot\\ x15\,(n+1) \end{array}\right| \begin{array}{c} =\\=\\=\\=\\=\\=\\=\\ \\ \\=\\=\\=\\ \\ \\= \end{array} \left|\underbrace{\begin{array}{c} 0\\ X\,0\,(n)\\ X\,1\,(n)\\ X\,2\,(n)\\ X\,3\,(n)\\ X\,4\,(n)\\ X\,3\,(n)\\ \cdot\\ \cdot\\ X10\,(n)\\ X11\,(n)\\ X12\,(n)\\ \cdot\\ \cdot\\ X14\,(n) \end{array}}_{\text{one shift}} \begin{array}{cccc} \oplus & X15\,(n) & \oplus & D\,(n)\\ \oplus & 0 & \oplus & 0\\ \oplus & 0 & \oplus & 0\\ \oplus & 0 & \oplus & 0\\ \oplus & 0 & \oplus & 0\\ \oplus & X15\,(n) & \oplus & D\,(n)\\ \oplus & 0 & \oplus & 0\\ & \cdot & & \cdot\\ & \cdot & & \cdot\\ \oplus & 0 & \oplus & 0\\ \oplus & X15\,(n) & \oplus & D\,(n)\\ \oplus & 0 & \oplus & 0\\ & \cdot & & \cdot\\ & \cdot & & \cdot\\ \oplus & 0 & \oplus & 0 \end{array}\right| \begin{array}{l} \leftarrow \text{feedback}\\ \\ \\ \\ \\ \leftarrow \text{feedback}\\ \\ \\ \\ \\ \leftarrow \text{feedback}\\ \\ \\ \\ \\ \end{array}$$

equation (1)

OR

$$\left|\begin{array}{c} X\,0\,(n+1)\\ X\,1\,(n+1)\\ \cdot\\ \cdot\\ \cdot\\ X15\,(n+1) \end{array}\right| \begin{array}{c} =\\=\\ \\ \\ \\= \end{array} \underbrace{\left|\begin{array}{c|c} 0\ \ldots & 0\\ \hline & 0\\ & \cdot\\ I & \cdot\\ & \cdot\\ & 0 \end{array}\right| * \left|\begin{array}{c} X\,0\,(n)\\ X\,1\,(n)\\ \cdot\\ \cdot\\ \cdot\\ X15\,(n) \end{array}\right|}_{\text{one shift}} \underbrace{\oplus\, X15\,(n) * \left|\begin{array}{c} a0\\ a1\\ \cdot\\ \cdot\\ \cdot\\ a15 \end{array}\right| \oplus\, D\,(n) * \left|\begin{array}{c} a0\\ a1\\ \cdot\\ \cdot\\ \cdot\\ a15 \end{array}\right|}_{\text{feedbacks}}$$

equation (2)

OR it can be written in another form:

$$\left|\begin{array}{c} X\,0\,(n+1)\\ X\,1\,(n+1)\\ \cdot\\ \cdot\\ \cdot\\ X15\,(n+1) \end{array}\right| \begin{array}{c} =\\=\\ \\ \\ \\= \end{array} \underbrace{\left|\begin{array}{c|c} 0 & a0\\ \hline & a1\\ & \cdot\\ I & \cdot\\ & \cdot\\ & a15 \end{array}\right|}_{\text{One shift and feedback from X15}} * \left|\begin{array}{c} X\,0\,(n)\\ X\,1\,(n)\\ \cdot\\ \cdot\\ \cdot\\ X15\,(n) \end{array}\right| + D\,(n) * \underbrace{\left|\begin{array}{c} a0\\ a1\\ \cdot\\ \cdot\\ \cdot\\ a15 \end{array}\right|}_{\text{Feedback from D(n)}}$$

$$\text{OR} \quad X\,(n+1) = T * X\,(n) \oplus D\,(n) * a$$

equation (3)

are no preceeding register stage. The elements a0, ..., a15 represent the corresponding feedback postions in the registers X0, ..., X15. For each register, X0, X5, and X12, where there is feedback, the corresponding elements, a0 = a5 = a12 = 1. For the other registers that do not have feedback, ai = 0. The same equation, (1) and (2), can yet be written in another form, equation (3). In this form, the identity matrix in T represents the shift, and the ai elements represent the feedback positions.

To construct the equations for the 8-bit look-ahead circuitry, it is required to shift eight data bits, D (n), where n = 0, ..., 7, into the LFSR. These eight shifts can be performed mathematically as follows:

$$
\begin{array}{llll}
\text{1st shift,} & X(n+1) & = & T * X(n) \oplus D(0) * a \\
\text{2nd shift,} & X(n+2) & = & T * X(n+1) \oplus D(1) * a \\
 & & = & T^2 * X(n) \oplus T * D(0) * a \oplus D(1) * a \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\text{8th shift,} & X(n+8) & = & T * X(n+7) \\
 & & = & T^8 * X(n) \oplus D(7) * a \\
 & & & \oplus T^7 * D(0) * a \oplus T^6 * D(1) * a \oplus T^5 \\
 & & & \quad * D(2) * a \oplus T^4 * D(3) * a \\
 & & & \oplus T^3 * D(4) * a \oplus T^2 * D(5) * a \oplus T^1 \\
 & & & \quad * D(6) * a \oplus \quad D(7) * a
\end{array}
$$

equation (4)

The contents of the registers after eight shifts can be found by solving equation (4). It is not suggested to attempt to solve these equations by hand! The results of this equation is the same as equation (0). The equations required for any M-bit look-ahead circuitry can be calculated in a similar way if the structure of the LFSR is put in the matrix form in equation (3).

Appendix B

Using PAL devices to implement Serial CRC Example, Serial 16-bit CRC, CRC-16

The procedures of implementing a serial N-bit CRC using PAL devices, can be summarised below:

(i) select the generator polynominal, G(X);
(ii) determine the initial condition for the LFSR, and whether to transmit the inverting or non-inverting redundancy check bits;
(iii) construct the required N-bit LFSR and the error flag;
(iv) partition the LFSR and error flag into the required PAL devices in a similar way as shown in Figures 5.0-1;
(v) Write the corresponding PAL design equations.

In the serial CRC design example, it is desired to implement the 16-bit serial CRC using the CRC-16 standard generator polynomial, where,

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

It is also desired to initialize the registers to all ONEs and transmit the non-inverting redundancy check bits.

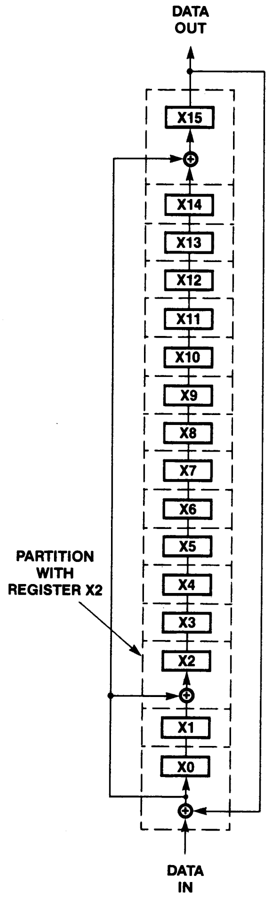


Figure 7.0. 16-Bit Linear Feedback Shift Register (LFSR) Implementing the CRC-16 Standard Generator Polynomial

The N-bit LFSR can be constructed easily from the generator polynomial, G(X). The required LFSR has sixteen registers, X0, ..., X15, with the feedback term (X15 ⊕ DIN), at registers X0, X2, and X15, as shown in Figure 7.0. The N-bit CRC required can be partitioned as shown in Figure 5.0.

The complete design for this 16-bit CRC is shown in Figure 7.1. Two pins, $\overline{\text{INIT}}$ and $\overline{\text{CONTRL}}$ are added for initializing the registers and controlling the feedback term respectively. When $\overline{\text{INIT}}$ is LOW, all registers are initialized to all ONEs. When $\overline{\text{CONTRL}}$ is LOW, the feedback terms to the registers are disabled so that the contents (redundancy check bits) residing in the registers can be shifted out undisturbed. PAL design specifications for this example are included in the last section of this appendix. In order to understand these specifications, consider the circuitry partitioned with register X2 shown in Figure 7.0. This circuitry consists of register X2 with a feedback term. The equation that describes this circuitry is:

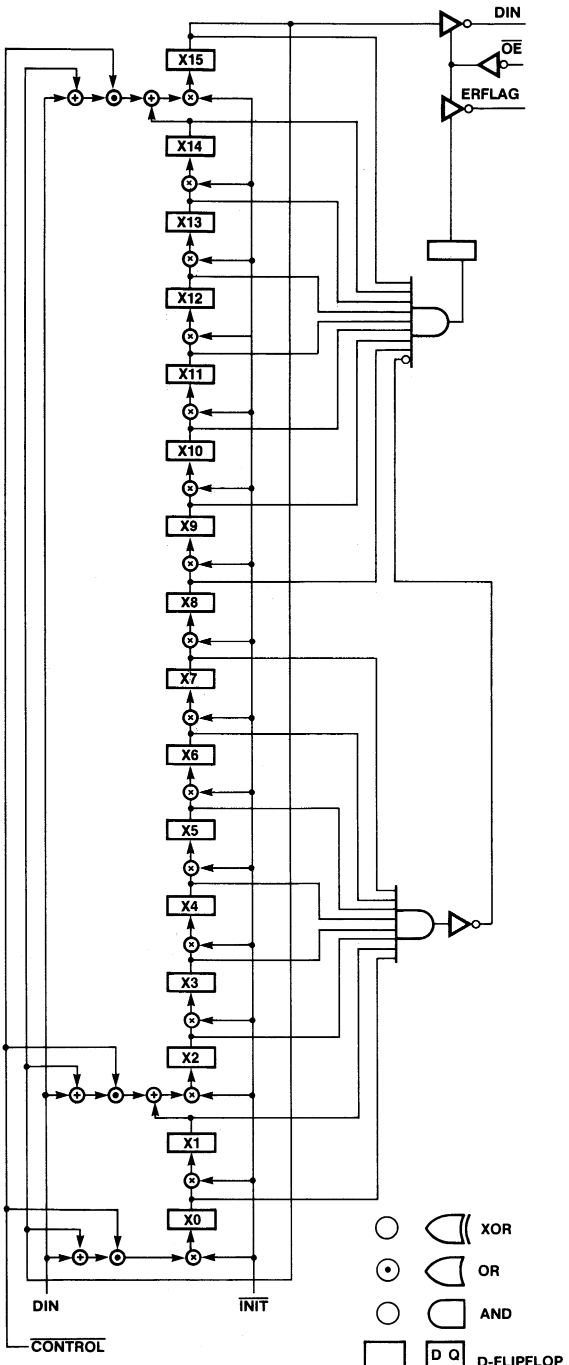


Figure 7.1. Logic Implementation (Simplified) for Serial 16-Bit CRC Using the CRC-16 Standard Generator Polynomial

$$X2 = \underbrace{(X15 \oplus DIN)}_{\text{feedback term}} \oplus \underbrace{X1}_{\text{previous register stage}}$$

Since the outputs of the PAL devices have naturally inverted outputs, this equation should be rewritten using DeMorgan's Theorem in order to get non-inverting outputs. Recall that it is desired to transmit non-inverting redundancy check bits in this design example. The rewritten equation is:

$\overline{X2} = (X15 \oplus DIN) \oplus \overline{X1}$

In order to control the feedback term, the feedback term is ANDed with $\overline{\text{CONTROL}}$ so that when $\overline{\text{CONTROL}}$ is set to LOW, the feedback term will be disabled.

$\overline{X2} = ((X15 \oplus DIN)^{*}\, \overline{\text{CONTROL}}) \oplus \overline{X1}$

$\overline{\text{INIT}}$ is ANDed with the feedback term as well as the $\overline{X1}$ term so that when $\overline{\text{INIT}}$ is set to LOW, $\overline{X2}$ will be zero, or register X2 will be one, shown below:

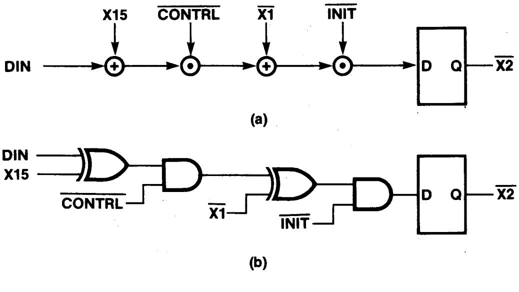
$\overline{X2} = (((X15 \oplus DIN)^{*}\, \overline{\text{CONTROL}}) \oplus \overline{X1})^{*}\, \overline{\text{INIT}}$ ....(1)

This equation describes the circuitry enclosed by partition X2 as shown in Figures 7.1 and 7.2a. This circuitry can be redrawn in another way as shown in Figure 7.2b. Equation (1) can be rewritten in another form:

$$\begin{aligned}\overline{X2} = {} & \overline{X1} * \overline{\text{INIT}} \\ & \oplus (\overline{X15} * \text{DIN} * \overline{\text{CONTRL}} * \overline{\text{INIT}} \\ & + \overline{X15} * \overline{\text{DIN}} * \overline{\text{CONTRL}} * \overline{\text{INIT}})\end{aligned}$$

Or using PAL design specifications with PAL20x devices,

$$\begin{aligned}& \overline{X2} : = \overline{X1} * \overline{\text{INIT}} \\ & + \overline{X1} * \overline{\text{INIT}} \\ & {:}{+}{:}\ \overline{X15} * \text{DIN} * \overline{\text{CONTROL}} * \overline{\text{INIT}} \\ & + \overline{X15} * \overline{\text{DIN}} * \overline{\text{CONTROL}} * \overline{\text{INIT}} \qquad ....(2)\end{aligned}$$



(a)

(b)

Figure 7.2. The Partition with Registor X2

$$\begin{aligned}\overline{X2} = {} & \overline{X1} * \overline{\text{INIT}} \\ & + (\overline{X15} * \overline{\text{DIN}} * \overline{\text{CONTRL}} * \overline{\text{INIT}} \\ & + \overline{X15} * \overline{\text{DIN}} * \overline{\text{CONTRL}} * \overline{\text{INIT}})\end{aligned}$$

The logic implementation of equation (1) using PAL design specifications in equation (2) is shown in Figure 7.3. The rest of the registers are implemented in a similar way.

When the redundancy check bits are transmitted directly, the contents residing inside the registers, when there is no error during transmission, should be all ZEROs. The equation for the error flag is simply:

$$\overline{\text{ERFLAG}} = (\overline{\text{X1}} * \overline{\text{X2}} *...* \overline{\text{X7}}) * (\overline{\text{X8}} *...* \overline{\text{X14}} * \overline{\text{X15}})$$

Using the partitioning in Figure 5.1, the equation can be further separated into two error flag equations, where

$$\overline{\text{ERFLAG0}} = \overline{\text{X1}} * \overline{\text{X2}} *...* \overline{\text{X7}}$$
$$\overline{\text{ERFLAG}} = \overline{\text{ERFLAG0}} * \overline{\text{X8}} * \overline{\text{X9}} *...* \overline{\text{X15}}$$

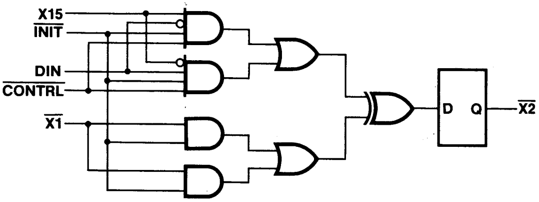


Figure 7.3. The Same Circuit with PAL design Specifications

$$\begin{aligned}\overline{\text{X2}} :=\ & \overline{\text{X1}} * \overline{\text{INIT}}\\ +\ & \overline{\text{X1}} * \overline{\text{INIT}}\\ :+:\ & \overline{\text{X15}} * \overline{\text{DIN}} * \overline{\text{CONTRL}} * \overline{\text{INIT}}\\ +\ & \overline{\text{X15}} * \overline{\text{DIN}} * \overline{\text{CONTROL}} * \overline{\text{INIT}}\end{aligned}$$

$\overline{\text{ERFLAG0}}$ is the intermediate error flag put into the PAL20x8, whereas $\overline{\text{ERFLAG}}$ is the final latched error flag put into the PAL20x10. The PAL design specifications of the above equations are:

$$\overline{\text{ERFLAG0}} = \overline{\text{X1}} * \overline{\text{X2}} *...* \overline{\text{X7}}$$
$$\begin{aligned}\overline{\text{ERFLAG}} :=\ & \overline{\text{ERFLAG0}} * \overline{\text{X8}} * \overline{\text{X9}} *...* \overline{\text{X15}} * \overline{\text{INIT}}\\ & + \overline{\text{INIT}}\end{aligned}$$

The $\overline{\text{INIT}}$ pin is ANDed with the error flag to initialize the error flag to ONE, which is the no error condition.

Serial 16-Bit CRC Generator (Using CRC-16 Standard)

Description

The serial 16-bit CRC generator will generate the sixteen redundancy check bits serially for transmission. It can also detect error during data reception by indicating error with an error flag, ERFLAG. This Serial 16-bit CRC chip implements the CRC-16 standard generator polynomial, where

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

The maximum input to output pin delay is 60ns, therefore, the maximum speed is about 16 MHz.

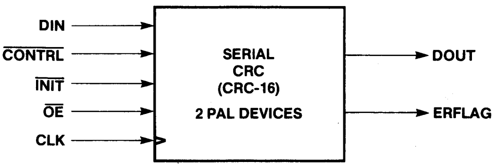


Figure 7.4. PAL Device Implementation of a Serial CRC Chip Using the Standard CRC-16 Generator Polynomial

To generate the redundancy check bits for transmission, the registers should be first initialized to all HIGH by setting the $\overline{\text{INIT}}$ pin to LOW on the first clock. Then data bits are clocked in bit by bit through DIN. After all the data bits are clocked in, the first redundancy check bits appears at the output of register X15. $\overline{\text{CONTROL}}$, at this point, should be set LOW in order to shift out the rest of the fifteen redundancy check bits. $\overline{\text{OE}}$ should only be set LOW during the whole period when the redundancy check bits are shifting out of the chip, whereas $\overline{\text{CONTROL}}$ should be set LOW one clock period after $\overline{\text{OE}}$ is set LOW. $\overline{\text{OE}}$, $\overline{\text{INIT}}$ and $\overline{\text{CONTROL}}$ should be set HIGH under normal operation. The error flag, $\overline{\text{ERFLAG}}$, will not be connected in this case. Note that if an external 2- to -1 mux is used to mux the data and the check bits, $\overline{\text{OE}}$ should leave LOW all the time to enable the outputs.

On the reception end, the registers should also be first initialized to all HIGH before any data bits is shifted in the chip. In this case, only the error flag will be connected. After all the transmitted data bits (including the sixteen redundancy check bits) are clocked in, on the next clock, $\overline{\text{ERFLAG}}$ will be LOW when there is no error, HIGH otherwise.

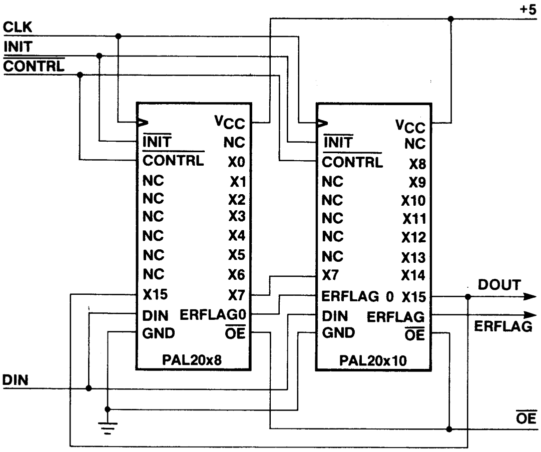


Figure 7.5. Diagram Showing How to Connect 2 PAL Devices to Implement the Serial CRC Generator Using the CRC-16 Standard

The circuitry for this serial 16-bit CRC generator is implemented with two PAL devices: PAL20x8 (Serial CRC, CRC-16, CHIP 1) and PAL20x10 (Serial CRC, CRC-16, CHIP 2). The connections of these PAL devices are shown in Figure 7.5.

```
PAL20X8                                       PAL SPECIFICATION
EXAMPLE, CRC-16                                  VIVIAN KONG
SERIAL CRC, CHIP 1                                 8/1/83
MMI, SANTA CLARA, CAFORNIA.
CLK /INIT /CONTRL NC NC NC NC NC NC X15 DIN GND
/OE ERFLAG0 X0 X1 X2 X3 X4 X5 X6 X7 NC VCC

/X0  :=    X15 *  DIN * /CONTRL * /INIT  ;X RAISED TO POWER 0
      +   /X15 * /DIN * /CONTRL * /INIT  ;COEFFICIENT 1

/X1  :=   /X0 * /INIT                    ;X RAISED TO POWER 1
      +   /X0 * /INIT                    ;COEFFICIENT 0

/X2  :=   /X1 * /INIT                    ;X RAISED TO POWER 2
      +   /X1 * /INIT                    ;COEFFICIENT 1
     :+:  /X15 *  DIN * /CONTRL * /INIT
      +    X15 * /DIN * /CONTRL * /INIT

/X3  :=   /X2 * /INIT                    ;X RAISED TO POWER 3
      +   /X2 * /INIT                    ;COEFFICIENT 0

/X4  :=   /X3 * /INIT                    ;X RAISED TO POWER 4
      +   /X3 * /INIT                    ;COEFFICIENT 0

/X5  :=   /X4 * /INIT                    ;X RAISED TO POWER 5
      +   /X4 * /INIT                    ;COEFFICIENT 0

/X6  :=   /X5 * /INIT                    ;X RAISED TO POWER 6
      +   /X5 * /INIT                    ;COEFFICIENT 0

/X7  :=   /X6 * /INIT                    ;X RAISED TO POWER 7
      +   /X6 * /INIT                    ;COEFFICIENT 0

IF (VCC)  /ERFLAG0 = /X0*/X1*/X2*/X3*/X4*/X5*/X6*/X7

                                         ;INTERMMEDIATE
                                         ;ERROR FLAG
```

```
FUNCTION TABLE

CLK /OE /INIT /CONTRL DIN X15 X0 X1 X2 X3 X4 X5 X6 X7 ERFLAG0
-------------------------------------------------------------
; / / /
;C O I C D X      X X X X X X X X    E
;      O                             R
;    N N                             F
;L       I 1                         L        COMMENTS
;    I T                             A
;      R                             G
;K E T L N 5      0 1 2 3 4 5 6 7    0
;------------------------------------------------------------
 X H X X X X      Z Z Z Z Z Z Z Z    X  OUTPUT DISABLED
 C L L H X X      H H H H H H H H    X  INIT TO ALL ONES
 C L H H L H      H H L H H H H H    X  SHIFT IN ZERO DATA
 C L H H L L      L H H L H H H H    X
 C L H H L H      H L L H L H H H    X
 C L H H L L      L H L L H L H H    X
 C L H H L H      H L L L L H L H    X
 C L H H L L      L H L L L L H L    X
 C L H H L H      H L L L L L L H    X
 C L H H L L      L H L L L L L L    X
 C L H H L H      H L L L L L L L    X
 C L H H L L      L H L L L L L L    X
 C L H H L H      H L L L L L L L    X
 C L H H L L      L H L L L L L L    X
 C L H H L H      H L L L L L L L    X
 C L H H L L      L H L L L L L L    X
 C L H H L L      L L H L L L L L    X
 C L H H L H      H L H H L L L L    X
 C L H H L H      H H H H H L L L    X SHIFT OUT CHECK BITS
 C X H L X X      H H H H H H L L    X 1
 C X H L X X      H H H H H H H L    X 2
 C X H L X X      H H H H H H H H    X 3
 C X H L X X      H H H H H H H H    X 4
 C X H L X X      H H H H H H H H    X 5
 C X H L X X      H H H H H H H H    X 6
 C X H L X X      H H H H H H H H    X 7
 C X H L X X      H H H H H H H H    X 8
 C X H L X X      H H H H H H H H    X 9
 C X H L X X      H H H H H H H H    X 10
 C X H L X X      H H H H H H H H    X 11
 C X H L X X      H H H H H H H H    X 12
 C X H L X X      H H H H H H H H    X 13
 C X H L X X      H H H H H H H H    X 14
 C X H L X X      H H H H H H H H    X 15-----------------
 C L L H X X      H H H H H H H H    X  INIT TO ALL ONES
```

```
C L H H L H      H H L H H H H H      X  SHIFT IN ZERO DATA
C L H H L L      L H H L H H H H      X
C L H H L H      H L L H L H H H      X
C L H H L L      L H L L H L H H      X
C L H H L H      H L L L L H L H      X
C L H H L L      L H L L L L H L      X
C L H H L H      H L L L L L L H      X
C L H H L L      L H L L L L L L      X
C L H H L H      H L L L L L L L      X
C L H H L L      L H L L L L L L      X
C L H H L H      H L L L L L L L      X
C L H H L L      L H L L L L L L      X
C L H H L H      H L L L L L L L      X
C L H H L L      L H L L L L L L      X
C L H H L L      L L H L L L L L      X
C L H H L H      H L H H L L L L      X
C L H H L H      H H H H H L L L      X  SHIFT IN CHECK BITS
C L H H H H      L H H H H H L L      X
C L H H L L      L L H H H H H L      X
C L H H L L      L L L H H H H H      X
C L H H L L      L L L L H H H H      X
C L H H L L      L L L L L H H H      X
C L H H L L      L L L L L L H H      X
C L H H L L      L L L L L L L H      X
C L H H L L      L L L L L L L L      X
C L H H L L      L L L L L L L L      X
C L H H L L      L L L L L L L L      X
C L H H L L      L L L L L L L L      X
C L H H H H      L L L L L L L L      X
C L H H H H      L L L L L L L L      X
C L H H H H      L L L L L L L L      X
C L H H H H      L L L L L L L L      X
C L H H H H      L L L L L L L L      X  ZERO CONTENTS
C L H H X X      X X X X X X X X      L  NO ERROR
-------------------------------------------------------------------
```

Description

This chip is the first part of the serial 16-bit CRC generator implementing the standard CRC-16 generator polynomial. It implements the logics for the eight registers, X0,..., X7, and also generates an intermediate error flag. ERFALG0. The inputs to this chip are DIN, $\overline{\text{INIT}}$, $\overline{\text{CONTRL}}$, and $\overline{\text{OE}}$.

```
PAL20X10                                      PAL SPECIFICATION
EXAMPLE, CRC-16                                 VIVIAN KONG
SERIAL CRC, CHIP 2                               8/1 /83
MMI, SANTA CLARA, CAFORNIA
CLK /INIT /CONTRL NC NC NC NC NC ERFLAG0 X7 DIN GND
/OE ERFLAG X8 X9 X10 X11 X12 X13 X14 X15 NC VCC

/X8   :=  /X7 * /INIT                        ;X RAISED TO POWER 8
       +  /X7 * /INIT                        ;COEFFICIENT 0

/X9   :=  /X8 * /INIT                        ;X RAISED TO POWER 9
       +  /X8 * /INIT                        ;COEFFICIENT 0

/X10  :=  /X9 * /INIT                        ;X RAISED TO POWER 10
       +  /X9 * /INIT                        ;COEFFICIENT 0

/X11  :=  /X10 * /INIT                       ;X RAISED TO POWER 11
       +  /X10 * /INIT                       ;COEFFICIENT 0

/X12  :=  /X11 * /INIT                       ;X RAISED TO POWER 12
       +  /X11 * /INIT                       ;COEFFICIENT 0

/X13  :=  /X12 * /INIT                       ;X RAISED TO POWER 13
       +  /X12 * /INIT                       ;COEFFICIENT 0

/X14  :=  /X13 * /INIT                       ;X RAISED TO POWER 14
       +  /X13 * /INIT                       ;COEFFICIENT 0

/X15  :=  /X14 * /INIT                       ;X RAISED TO POWER 15
       +  /X14 * /INIT                       ;COEFFICIENT 1
      :+: /X15 * DIN * /CONTRL * /INIT
       +   X15 */DIN * /CONTRL * /INIT

/ERFLAG :=  /ERFLAG0*/X8*/X9*/X10*/X11*/X12*/X13*/X14*/X15*/INIT
         +  INIT
                                             ;FINAL ERROR FLAG
```

```
FUNCTION TABLE
CLK /OE /INIT /CONTRL ERFLAG0 DIN X7
X8 X9 X10 X11 X12 X13 X14 X15 ERFLAG
;-----------------------------------------------------------------
;  / / /
;C O I C E D X    X X X X X X X X     E
;      O R                            R
;    N N F                            F
;L       L I          1 1 1 1 1 1     L       COMMENTS
;    I T A                            A
;      R F                            G
;K E T L 0 N 7    8 9 0 1 2 3 4 5
------------------------------------------------------------------
 X H X X X X X    Z Z Z Z Z Z Z Z     Z   OUTPUT DISABLED
 C L L X X X X    H H H H H H H H     L   INIT TO ALL ONES
 C L H H X L H    H H H H H H H L     X   GENERATING REDUNDANCY
 C L H H X L H    H H H H H H H H     X   CHECK BITS
 C L H H X L H    H H H H H H H L     X
 C L H H X L H    H H H H H H H H     X
 C L H H X L H    H H H H H H H L     X
 C L H H X L H    H H H H H H H H     X
 C L H H X L L    L H H H H H H L     X
 C L H H X L H    H L H H H H H H     X
 C L H H X L L    L H L H H H H L     X
 C L H H X L L    L L H L H H H H     X
 C L H H X L L    L L L H L H H L     X
 C L H H X L L    L L L L H L H H     X
 C L H H X L L    L L L L L H L L     X
 C L H H X L L    L L L L L L H L     X
 C L H H X L L    L L L L L L L H     X
 C L H H X L L    L L L L L L L H     X
 C L H H X L L    L L L L L L L H     X   SHIFT OUT CHECK BITS
 C L H L X X L    L L L L L L L L     X   1
 C L H L X X L    L L L L L L L L     X   2
 C L H L X X L    L L L L L L L L     X   3
 C L H L X X H    H L L L L L L L     X   4
 C L H L X X H    H H L L L L L L     X   5
 C L H L X X H    H H H L L L L L     X   6
 C L H L X X H    H H H H L L L L     X   7
 C L H L X X H    H H H H H L L L     X   8
 C L H L X X H    H H H H H H L L     X   9
 C L H L X X H    H H H H H H H L     X   10
 C L H L X X H    H H H H H H H H     X   11
 C L H L X X H    H H H H H H H H     X   12
 C L H L X X H    H H H H H H H H     X   13
 C L H L X X H    H H H H H H H H     X   14
 C L H L X X H    H H H H H H H H     X   15--------------
```

```
C L L X X X X   H H H H H H H H   L   INIT TO ALL ONES
C L H H X L H   H H H H H H H L   X   DATA RECEPTION
C L H H X L H   H H H H H H H H   X   MESSAGE AND
C L H H X L H   H H H H H H H L   X   THE REMAINDER
C L H H X L H   H H H H H H H H   X
C L H H X L H   H H H H H H H L   X
C L H H X L H   H H H H H H H H   X
C L H H X L L   L H H H H H H L   X
C L H H X L H   H L H H H H H H   X
C L H H X L L   L H L H H H H L   X
C L H H X L L   L L H L H H H H   X
C L H H X L L   L L L H L H H L   X
C L H H X L L   L L L L H L H H   X
C L H H X L L   L L L L L H L L   X
C L H H X L L   L L L L L L H L   X
C L H H X L L   L L L L L L L H   X
C L H H X L L   L L L L L L L H   X
C L H H X L L   L L L L L L L H   X
C L H H X H L   L L L L L L L L   X
C L H H X L L   L L L L L L L L   X
C L H H X L L   L L L L L L L L   X
C L H H X L H   H L L L L L L L   X
C L H H X L H   H H L L L L L L   X
C L H H X L H   H H H L L L L L   X
C L H H X L H   H H H H L L L L   X
C L H H X L H   H H H H H L L L   X
C L H H X L L   L H H H H H L L   X
C L H H X L L   L L H H H H H L   X
C L H H X L L   L L L H H H H H   X
C L H H X H L   L L L L H H H H   X
C L H H X H L   L L L L L H H H   X
C L H H X H L   L L L L L L H H   X
C L H H X H L   L L L L L L L H   X
C L H H L H L   L L L L L L L L   X   ZERO CONTENTS
C L H H X X X   X X X X X X X X   L   NO ERROR
-----------------------------------------------------------------
```

Description

This chip is the second part of the serial 16-bit CRC generator implementing the standard CRC-16 generator polynomial. It implements the logics required for the eight registers, X8, ..., X15, and it also generates the final error flag, ERFLAG, for error detection. The redundancy check bits will be shifted out at the output of register X15. The inputs to this chip are CLK, $\overline{\text{OE}}$, DIN $\overline{\text{CONTRL}}$ and $\overline{\text{INIT}}$.

Appendix C

Using PAL Devices to Implement Parallel CRC, Example, 8-Bit Parallel CRC Chip (Using CRC-CCITT)

Although there is no standard way of partitioning an M-bit CRC in the PAL devices, there is a design procedure. The procedure of designing an M-bit parallel CRC generator can be summarized as follows:

(i) select the CRC generator polynomial, G(X);

(ii) calculate the M-bit look-ahead equations, using the methods described in Appendix A, for different M that might fit your application;

(iii) compare the complexity of these equations and pick the most desired set;

(iv) partition the equations in a similar way as in the 8-bit parallel CRC design example.

Description

The 8-bit parallel CRC generator performs CRC in a more efficient, parallel fashion. The maximum input to output pin delay is 90ns, therefore the maximum speed is 10MHz. This chip is implemented with the CRC-CCITT generator polynomial G(X), where

$$G(X) = X^{16} + X^{12} + X^{5} + 1$$

It is designed to interface with either an 8-bit data bus or 16-bit bus. This CRC chip will take eight bits of data in parallel and generate sixteen redundancy check bits in parallel. Therefore, in the case of an 8-bit bus, a 16- to -8 mux is required to multiplex the upper and lower eight bits of the sixteen redundancy check bits. As for interfacing with a 16-bit bus, a 16-to-8 mux might be necessary to multiplex the 16-bit data before feeding into the CRC chip (see Figures 8.6 - 8.9).

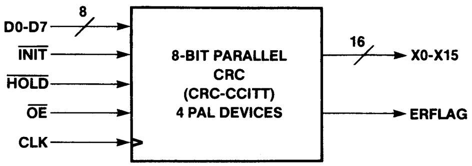


Figure 8.0. PAL Device Implementation of an 8-Bit Parallel CRC Chip Using the CRC-CCITT Standard

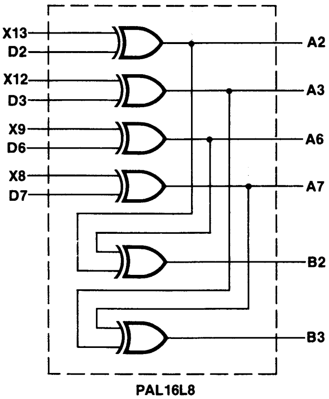


Figure 8.1. 8-Bit Parallel CRC, Chip 1

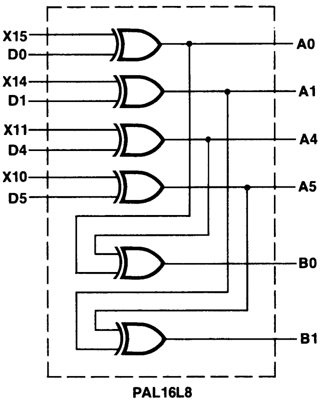


Figure 8.2. 8-Bit Parallel CRC, Chip 2

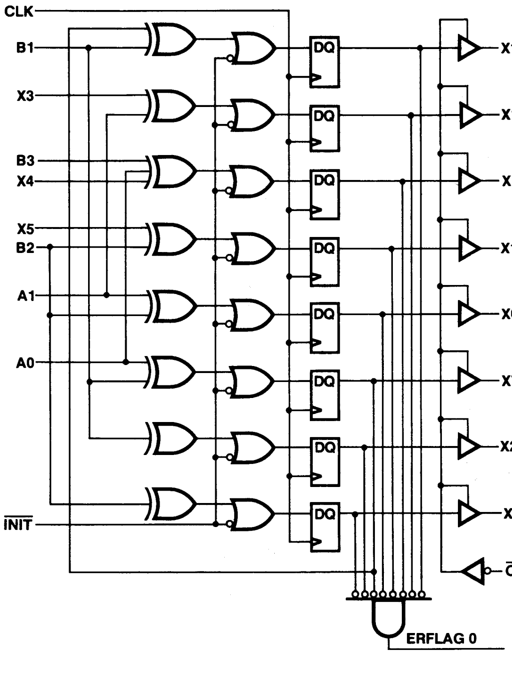


Figure 8.3. 8-Bit Parallel CRC, Chip 3

To generate the redundancy check bits for transmission, the registers should first be initialized to all HIGH by setting $\overline{\text{INIT}}$ to LOW on the first clock. Then $\overline{\text{INIT}}$ should be set to HIGH. The output enable pin, $\overline{\text{OE}}$, should be held HIGH to disable the

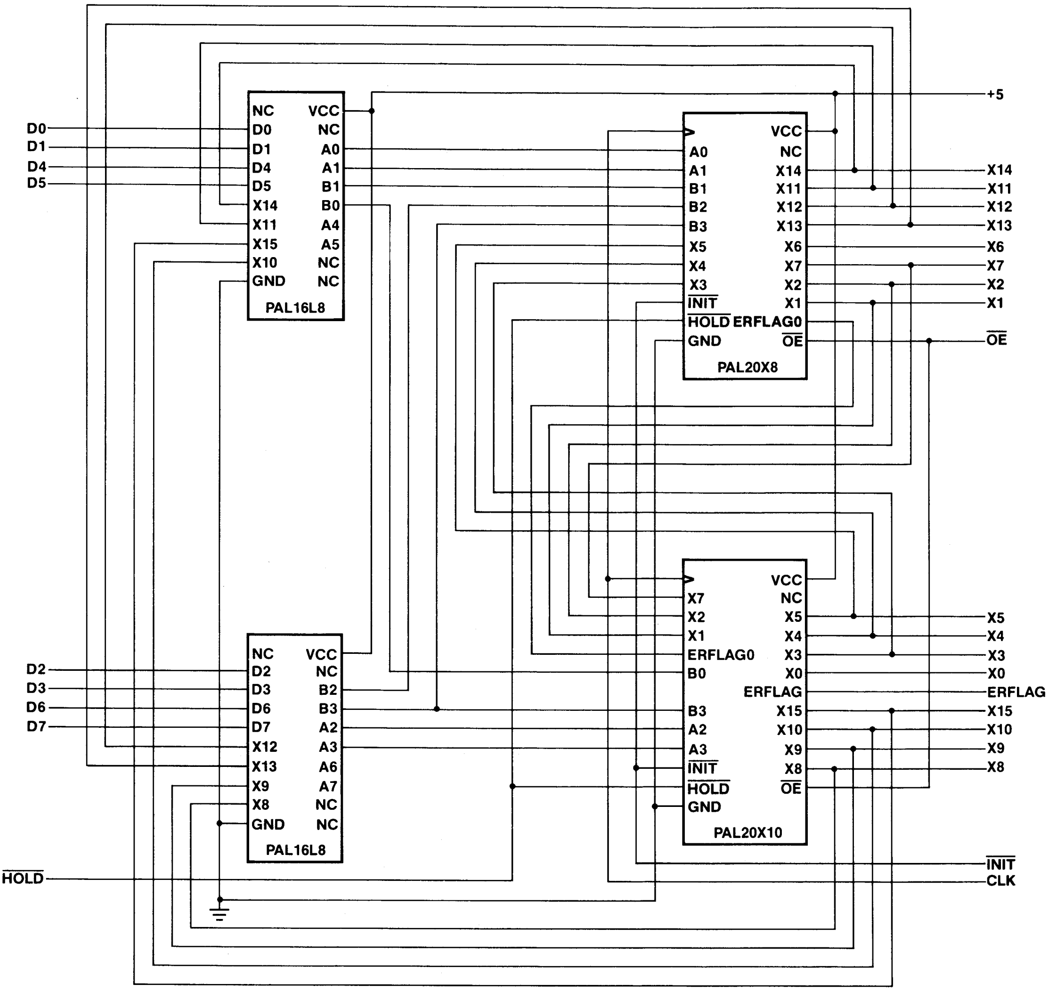


Figure 8.5. Diagram Showing How to Connect 4 PAL devices to Implement the 8-Bit Parallel CRC Chip

outputs when clocking in the 8-bit data bytes. After the last byte has been clocked in, $\overline{\text{OE}}$ should be held LOW to enable the outputs for transmitting the sixteen CRC bits. The upper byte of the sixteen CRC bits is first muxed out. Then $\overline{\text{HOLD}}$ should be enabled to hold the CRC bits for one more clock in order to mux out the lower byte. On data reception, the registers should also first be initialized. After all the transmitted data, including the CRC bits, have been clocked in the chip, ERFLAG should then be checked for any errors.

The 8-bit parallel CRC-CCITT chip consists of four PAL devices, two PAL16L8 devices, and two PAL20x10 devices. The chip basically implements equation (0) derived in Appendix A. The group of equations is partitioned into subgroups, which are put into the four PAL devices respectively, as shown in Figures 8.1 - 8.4. The interconnections are shown in Figure 8.5.

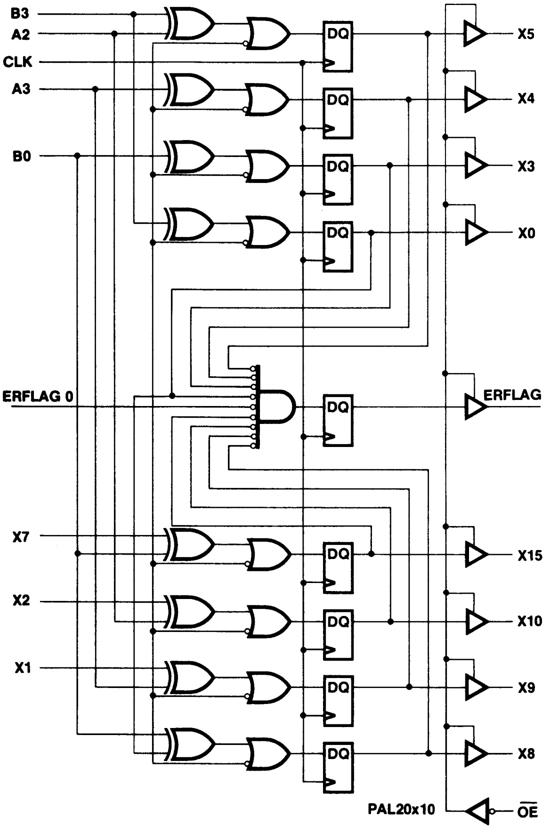


Figure 8.4. 8-Bit Parallel CRC, Chip 4 (for Simplicity, HOLD is not Included in the Diagram)

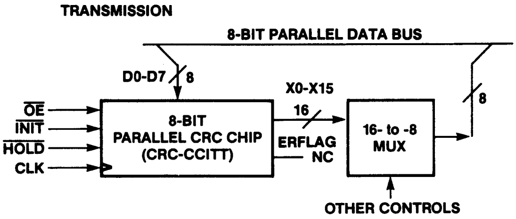


Figure 8.6. Interfacing the 8-Bit Parallel CRC Chip with an 8-Bit Bus

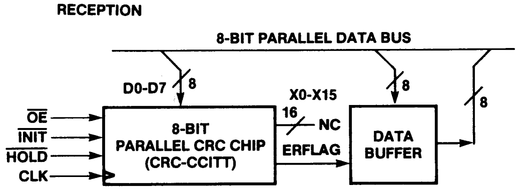


Figure 8.7. Interfacing the 8-Bit Parallel CRC with an 8-Bit Bus

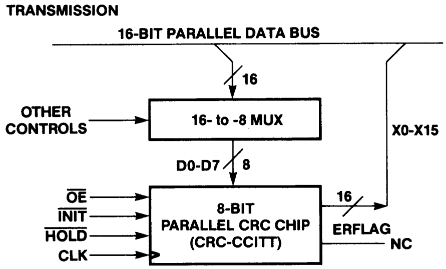


Figure 8.8 Interfacing the 8-Bit Parallel CRC Chip with a 16-Bit Bus

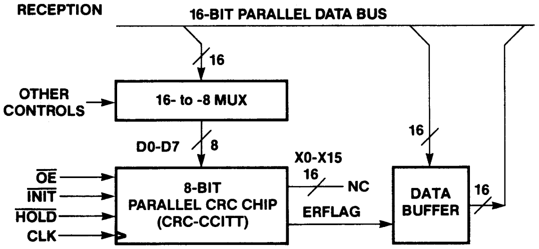


Figure 8.9. Interfacing the 8-Bit Parallel CRC Chip with a 16-Bit Bus

```
PAL16L8                                 PAL SPECIFICATIONS
PARALLEL BUS                            Vivian Kong
PARALLEL CRC (CRC-CCITT) CHIP 1          7/21/83
MMI,SANTA CLARA,CALIFORNIA
NC D0 D1 D4 D5 X14 X11 X15 X10 GND
NC NC A5 A4 B0 B1 A1 A0 NC VCC
IF (VCC) /A0  =  D0 * X15
              + /D0 */X15

IF (VCC) /A1  =  D1 * X14
              + /D1 */X14

IF (VCC) /A4  =  D4 * X11
              + /D4 */X11

IF (VCC) /A5  =  D5 * X10
              + /D5 */X10

IF (VCC) /B0  =  A0 * A4
              + /A0 */A4

IF (VCC) /B1  =  A1 * A5
              + /A1 */A5

FUNCTION TABLE

X10 X11 X14 X15 D0 D1 D4 D5
A0 A1 A4 A5 B0 B1

; X X X X D D D D   A A A A B B
; 1 1 1 1                               COMMENTS
; 0 1 4 5 0 1 4 5   0 1 4 5 0 1
-------------------------------------------------------
  H H H H L L L L   H H H H L L  MESSAGE WITH ALL
  L L H H L L L L   H H L L H H  ZEROS
  H H L L L L L L   L L H H H H
  H H H H L L L L   H H H H L L
  H L L H H L L H   L L L L L L
  L L H H H H L L   L L L L L L--------------------
  H H H H L L L L   H H H H L L  SAME MESSAGE BUT
  L L H H L L H L   H H H L L H  WITH ONE BIT IN
  H H L H L L L L   H L H H L H  ERROR
  H L H L L L L L   L H L H L L
  H H L L H L L H   H L H L L L
  H L H L H H L L   H L L H H H
-------------------------------------------------------
```

Description

This chip is the first part of the 8-bit Parallel CRC generator using the CRC-CCITT standard generator polynomial. This chip will generate two level XORs. The first level the output terms A0, A1, A4, and A5, the second level the output terms B0, and B1. The inputs to this chip are data bits D0, D1, D4, D5, and the outputs of the registers X10, X11, X14 and X15.

```
PAL16L8                                 PAL SPECIFICATIONS
8-BIT PARALLEL BUS                      Vivian Kong
PARALLEL CRC (CRC-CCITT) CHIP 2          7/21/83.
MMI, SANTA CLARA, CALIFORNIA
NC D2 D3 D6 D7 X12 X13 X9 X8 GND
NC NC A7 A6 A3 A2 B3 B2 NC VCC

IF (VCC) /A2 =  D2 *  X13
             + /D2 * /X13

IF (VCC) /A3 =  D3 *  X12
             + /D3 * /X12

IF (VCC) /A6 =  D6 *  X9
             + /D6 * /X9

IF (VCC) /A7 =  D7 *  X8
             + /D7 * /X8

IF (VCC) /B2 =  A2 *  A6
             + /A2 * /A6

IF (VCC) /B3 =  A3 *  A7
             + /A3 * /A7

FUNCTION TABLE

X8 X9 X12 X13 D2 D3 D6 D7
A2 A3 A6 A7 B2 B3

; X X X X D D D D   A A A A B B
;     1 1                               COMMENTS
; 8 9 2 3 2 3 6 7   2 3 6 7 2 3
-----------------------------------------------------
  H H H H L L L L   H H H H L L
  H L L H L L L L   H L L H H H  MESSAGE WITH ALL
  H L H L L L L L   L H L H L L  ZEROS
  L L L L L L L L   L L L L L L
  L L L L L L L L   L L L L L L
  L L L L L L L L   L L L L L L-------------------
  H H H H L L L L   H H H H L L
  H L L H L L L L   H L L H H H  SAME MESSAGE BUT
  L L H L L L L L   L H L L L H  WITH ONE BIT IN
  H L L L L L L L   L L L H L H  ERROR
  H L L H L L L L   H L L H H H
  H L L L L L L L   L L L H L H
-----------------------------------------------------
```

Description

This chip is the second part of the 8-bit Parallel CRC generator using the CRC-CCITT standard. This chip will generate two level XORs. The first level the output terms A0, A1, A4, and A5, the second level the output terms B2, and B3. The inputs to this chip are data bits D2, D3, D6, D7, and the outputs of the registers X8, X9, X12 and X13.

```
PAL20X8                                        PAL SPECIFICATIONS
8-BIT PARALLEL BUS                             Vivian Kong
PARALLEL CRC (CRC-CCITT) CHIP 3                 7/21/83.
MMI, SANTA CLARA, CAFORNIA
CLK A0 A1 B1 B2 B3 X5 X4 X3 /INIT /HOLD GND
/OE ERFLAG0 X1 X2 X7 X6 X13 X12 X11 X14 NC VCC

/X1 := /B2 * /INIT * /HOLD                       ;REGISTER X1
     + /B2 * /INIT * /HOLD
    :+: /X1 * /INIT * HOLD

/X2 := /B1 * /INIT * /HOLD                       ;REGISTER X2
     + /B1 * /INIT * /HOLD
    :+: /X2 * /INIT * HOLD

/X6 :=  A1 * B2 * /INIT * /HOLD                  ;REGISTER X6
     + /A1 */B2 * /INIT * /HOLD
    :+: /X6 * /INIT* HOLD

/X7 :=  A0 * B1  * /INIT * /HOLD                 ;REGISTER X7
     + /A0 */B1  * /INIT * /HOLD
    :+: /X7 * /INIT * HOLD

/X11 :=  X3 *  A1 * /INIT * /HOLD              ;REGISTER X11
      + /X3 * /A1 * /INIT * /HOLD
    :+: /X11 * /INIT * HOLD

/X12 :=  A0 *  B3 * /INIT * /HOLD              ;REGISTER X12
      + /A0 * /B3 * /INIT * /HOLD
     :+: X4 * /INIT * /HOLD
      + /X12 * /INIT * HOLD

/X13 :=  X5 *  B2 * /INIT * /HOLD                ;REGISTER X13
      + /X5 * /B2 * /INIT * /HOLD
    :+: /X13 * /INIT * HOLD

/X14 :=  X6 *  B1 * /INIT * /HOLD                ;REGISTER X14
      + /X6 * /B1 * /INIT * /HOLD
    :+: /X14 * /INIT * HOLD

IF (VCC) /ERFLAG0 = /X1*/X2*/X6*/X7*/X11*/X12*/X13*/X14*/INIT
```

```
FUNCTION TABLE

CLK /OE /INIT /HOLD A0 A1 B1 B2 B3 X3 X4 X5
X1 X2 X6 X7 X11 X12 X13 X14 ERFLAG0

;  / / /
;C O I H A A B B B X X X      X X X X X X X X E
;                                             R
;    N O                                      F
;L                                    1 1 1 1 L      COMMENTS
;    I L                                      A
;                                             G
;K E T D 0 1 1 2 3 3 4 5      1 2 6 7 1 2 3 4 0
-----------------------------------------------------------------
 X H X X X X X X X X X X      Z Z Z Z Z Z Z Z X
 C L L X X X X X X X X X      H H H H H H H H X  INIT REGISTERS
 C L H H H H L L L H H H      L L H H L L H H X  MESSAGE
 C L H H H H H H H L H H      H H L L H H L L X
 C L H H L L H L L H L L      L H L H H L L H X
 C L H H H H L L L H H L      L L H H L L L L X
 C L H H L L L L L L L L      L L L L L L L H X  CRC BITS
 C L H H L L L L L L L L      L L L L L L L L X
 C L H L X X X X X X X X      L L L L L L L L L--NO ERROR-------
 C L L X X X X X X X X X      H H H H H H H H X  INIT REGISTERS
 C L H H H H L L L H H H      L L H H L L H H X  MESSAGE WITH
 C L H H H H H H H L H H      H H L L H H L L X  1 BIT ERROR
 C L H H H L H L H L L L      L H L L L L L H X
 C L H H L H L L H L H H      L L H L H L H L X
 C L H H H L L H H L L H      H L H H L L L H X  CRC BITS
 C L H H H L H L H L L L      L H L L L L L L X
 C L H L X X X X X X X X      L H L L L L L L H--ERROR----------
------------------------------------------------------------------
```

Description

This is the third part of the 8-bit Parallel CRC generator using the CRC-CCITT standard. This chip will generate the current outputs of the registers X1, X2, X6, X7, X11, X12, X13, X14 and the intermediate error flag, ERFLAG0. The inputs to this chip are XOR terms, A0, A1, B1, B2, B3, generated by the first and the second parts of this CRC chip, and the outputs of the registers X3, X4, X5 and X6.

```
PAL20X10                                        PAL SPECIFICATIONS
8-BIT PARALLEL BUS                              Vivian Kong
PARALLEL CRC (CRC-CCITT)  CHIP 4                 7/21/83.
MMI, SANTA CLARA, CALIFORNIA
CLK X7 X2 X1 ERFLAG0 B0 B3 A2 A3 /INIT /HOLD GND
/OE X8 X9 X10 X15 ERFLAG X0 X3 X4 X5 NC VCC

/X0 := /B3 * /INIT * /HOLD                      ;REGISTER X0
     + /B3 * /INIT * /HOLD
    :+: /X0 * /INIT * HOLD

/X3 := /B0 * /INIT * /HOLD                      ;REGISTER X3
     + /B0 * /INIT * /HOLD
    :+: /X3 * /INIT * HOLD

/X4 := /A3 * /INIT * /HOLD                      ;REGISTER X4
     + /A3 * /INIT * /HOLD
    :+: /X4 * /INIT * HOLD

/X5 :=  A2 *  B3 * /INIT * /HOLD                ;REGISTER X5
     + /A2 * /B3 * /INIT * /HOLD
    :+: /X5 * /INIT * HOLD

/X8 :=  X0 *  B0 * /INIT * /HOLD                ;REGISTER X8
     + /X0 * /B0 * /INIT * /HOLD
    :+: /X8 * /INIT * HOLD

/X9 :=  X1 *  A3 * /INIT * /HOLD                ;REGISTER X9
     + /X1 * /A3 * /INIT * /HOLD
    :+: /X9 * /INIT * HOLD

/X10 :=  X2 *  A2 * /INIT * /HOLD               ;REGISTER X10
      + /X2 * /A2 * /INIT * /HOLD
    :+: /X10 * /INIT * HOLD

/X15 :=  X7 *  B0 * /INIT * /HOLD               ;REGISTER X15
      + /X7 * /B0 * /INIT * /HOLD
    :+: /X15 * /INIT * HOLD

/ERFLAG := /X0*/X3*/X4*/X5*/X8*/X9*/X10*/X15*/ERFLAG0
         + INIT

                                    ;FINAL ERROR FLAG
```

```
FUNCTION TABLE

CLK /OE /INIT /HOLD A2 A3 B0 B3 X1 X2 X7 ERFLAG0
X0 X3 X4 X5 X8 X9 X10 X15 ERFLAG

;  / / /               /
;C O I H A A B B X X X E     X X X X X X X X E
;                      R                     R
;    N O               F                     F
;L                     L                 1 1 L   COMMENTS
;    I L               A                     A
;                      G                     G
;K E T D 2 3 0 3 1 2 7 0     0 3 4 5 8 9 0 5
-------------------------------------------------------------
 X H X X X X X X X X X X     Z Z Z Z Z Z Z Z Z  OUTPUT DISABLED
 C L L X X X X X X X X X     H H H H H H H H L  INIT TO HIGH
 C L H H H H L L H H H X     L L H H H L L H X  MESSAGE
 C L H H H L H H L L H X     H H L L H L H L X
 C L H H L H H L H H L X     L H H L L L H H X
 C L H H L L L L L H H X     L L L L L L H H X
 C L H H L L L L L L H X     L L L L L L L H X
 C L H H L L L L L L L X     L L L L L L L L X  CRC BITS
 C L H L X X X X X X X L     L L L L L L L L L--NO ERROR---------
 C L L X X X X X X X X X     H H H H H H H H L  INIT TO HIGH
 C L H H H H L L H H H X     L L H H H L L H X  MESSAGE WITH 1
 C L H H H L L H L L H X     H L L L L L H H X  BIT ERROR
 C L H H L H L H H H L X     H L H H H L H L X
 C L H H L L L H L H L X     H L L H H L H L X
 C L H H H L L H L L L X     H L L L H L H L X
 C L H H L L H H H L H X     H H L H L H L L X  CRC BITS
 C L H L X X X X X X X H     L H L H L H L L H--ERROR-------------
-------------------------------------------------------------
```

Description

This chip is the last part of the 8-bit Parallel CRC generator using the CRC-CCITT standard. This chip will generate the current outputs of the registers X0, X3, X4, X5, X8, X9, X10, and X15. The inputs to this chip are XOR terms, A2, A3, B0, B3, generated by the first and the second parts of this CRC chip, and also the previous output state of the registers X0, X1, X2 and X7.

Acknowledgements

I would like to thank all who helped in the preparation of this application note, especially Nadia Sachs, who provided me with valuable information on the subject of Cyclic Redundancy Check.

References

r1. Neal Glover, *"Practical Error Correction Design for Engineers,"* Data Systems Technology, Corp., 1982.

r2. Nadia Sachs, *"Cyclic Redundancy Check using PAL devices."* AN-105, Monolithic Memories Inc., 2175 Mission College Blvd, Santa Clara, CA 95050.

r3. James F. Nebus, *"Parallel Data Compression for fault Tolerance,"* Computer Design, April 5, 1983.

r4. Dieter Schoene and Rene H. Terlet, *"Disk File Error Correction with a PLA,"* Computer Design, March, 1982.

r5. Robert Swanson, *"Understanding Cyclic Redundancy Codes,"* Computer Design, November, 1975.

r6. Programming Array Logic Handbook, Monolithic Memories Inc., Monolithic Memories Inc., 2175 Mission College Blvd, Santa Clara, CA 95050.

68000 Interrupt Controller

Danesh Tavana

Abstract

Commercial and industrial microprocessor based systems consist of the basic block units of CPU, memory, and I/O devices. While executing the instructions in the memory the CPU must somehow be interrupted to service requests from various I/O devices. The 68000 microprocessor is a powerful 16-bit processor which makes provisions for 256 different interrupt routines. A simple and cost effective way of interfacing a peripheral's interrupt request signal to the CPU is through a Programmable Array Logic. This paper introduces two ways of designing such an interface with PALs.

Introduction

Commercial and industrial microprocessor based systems typically consist of a processor interfaced with many peripherals which randomly require service. To fully utilize the processor's computing potentials sources of hardware interrupt must be used to free the processor from software routines. The 68000 16-bit microprocessor has 256 different exception processing vectors which point to a predetermined location in the program memory space. There are 192 external user interrupt vectors and seven autovector interrupts. Table 1 shows the exception vector assignments. The interrupt structure of the 68000 will be discussed in more detail along with two methods of designing an interrupt controller using PAL device (Programmable Array Logic).

TABLE 1
EXCEPTION VECTOR ASSIGNMENT

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
–	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14*	56	038	SD	(Unassigned, reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16-23*	64	04C	SD	(Unassigned, reserved)
	95	05F		–
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	191	0BF		–
48-63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		–
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		–

*Vector numbers 12, 13, 14, 16 through 23 and 48 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.

68000 Exception Processing And Pin Description

The interrupt structure of the 68000 can grant up to 256 types of interrupt requests. These interrupts or exceptions are generated either by external or internal causes and are serviced by directing the flow of program to an exception processing routine. Table 1 lists these exceptions and their respective address in memory. Exception vectors are locations in memory where the processor fetches the address of a routine or subprogram which will service that exception. The exception vector is either internally or externally generated depending on the cause of the interrupt. The externally generated exceptions are the interrupts which are placed on the interrupt control pins ($\overline{\text{IPL2}}$, $\overline{\text{IPL1}}$, $\overline{\text{IPL0}}$), bus errors, and reset requests. The interrupts placed on control pins $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ are requests from peripheral devices. The internally generated exceptions come from instructions, or from address error or tracing. These different types of exceptions and their priority is shown in Table 2. We will focus on group 1 interrupt exceptions.

TABLE 2
EXCEPTION GROUPING AND PRIORITY

Group	Exception	Processing
0	Reset Bus Error Address Error	Exception processing begins within two clock cycles.
1	Trace **Interrupt** Illegal Privilege	Exception processing begins before the next instruction.
2	TRAP, TRAPV, CHK, Zero Divide	Exception processing is started by normal instruction execution.

The pinout of the 68000 is shown in Figure 1. The three interrupt control pins ($\overline{\text{IPL2}}$, $\overline{\text{IPL1}}$, $\overline{\text{IPL0}}$) are asynchronous active low inputs which indicate the encoded priority level of

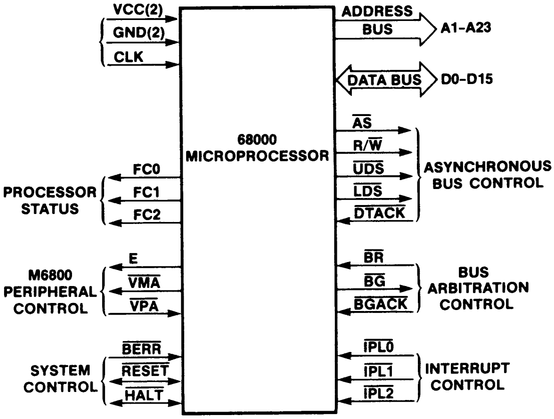


Figure 1

the device requesting an interrupt. The least significant bit is $\overline{\text{IPL0}}$ and the most significant bit is $\overline{\text{IPL2}}$. Level seven, ($\overline{\text{IPL2}}$, $\overline{\text{IPL1}}$, $\overline{\text{IPL0}}$) = (000), has the highest priority, while level zero, (111), indicates that no interrupts are requested. All lower or equal priority interrupts are masked during the interrupt service routine of the present interrupting device. There are two ways by which a peripheral device can interrupt the normal flow of program: autovectoring or external vector number generation. The function code pins (FC2, FC1, and FC0) all become high during an interrupt acknowledge cycle. The interrupt acknowledge cycle always follows an interrupt request only after the present instruction cycle is completed. Thus interrupt acknowlege cycles come in between the instruction cycles and no information is lost.

The signals which are used during an external interrupt request are listed below with a description of their behavior and function.

ADDRESS BUS: The 24-bit address bus holds the address of the data to be accessed. During an interrupt acknowledge cycle the lower three bits (A3 A2 A1) hold the encoded level of the interrupt being serviced. If a level 5 interrupt is being serviced then (A3 A2 A1) will be (101) respectively.

DATA BUS: The lower 8-bits of this 16-bit data bus must contain the vector number during a user interrupt acknowledge cycle. If an auto-vector routine is in process then the data bus is ignored.

$\overline{\text{AS}}$: The processor asserts address strobe anytime there is a valid data on the address bus. It remains asserted for as long as address is valid.

R/$\overline{\text{W}}$: This signal defines the data bus transfer as a read or a write cycle. During an interrupt acknowledge cycle the processor is in a read mode.

$\overline{\text{UDS}}$, $\overline{\text{LDS}}$: The upper & lower data strobes are asserted when the processor is in read or write instruction cycle. Upper strobe enables the most significant byte of data while the lower strobe enables the least significant.

$\overline{\text{DTACK}}$: Data transfer acknowledge is an externally generated signal which tells the processor that valid data is present on the data bus. If $\overline{\text{DTACK}}$ is not asserted before the falling edge of S4 (S4 is the fourth CPU clock state in a seven state instruction cycle) then wait states are introduced.

$\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$: The three interrupt control pins are inputs which contain the encoded priority level of the external interrupting device. Note that these are active low pins where (000) is level seven encoded.

FC2-FC0: Function code input pins indicate the processors status. When they are all high the processor is in an interrupt acknowledge cycle.

E,$\overline{\text{VMA}}$,$\overline{\text{VPA}}$: The enable, valid memory address, and valid peripheral address are the standard 6800 family signals. $\overline{\text{VPA}}$ is also used when auto-vectoring.

The following two design examples use PAL units as a simple and cost effective interrupt controller interface to the 68000. The first method introduces external vector generation while the second design example will show how auto-vectoring can be done on the 68000.

Prioritized Individually Vectored Interrupt (Method 1)

As shown in the interrupt acknowledge sequence flow chart and timing diagram (Fig. 2), a peripheral device must interrupt the processor by placing the encoded level of priority on the interrupt control pins ($\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$). The processor then grants the interrupt after the completion of the current instruction cycle. The encoded priority level of the interrupt is placed on address bus bits A3-A1 during the interrupt acknowledge cycle. The status code (FC2-FC0) become high which is an indication of an interrupt acknowledge cycle. Once the lower data strobe ($\overline{\text{LDS}}$) is asserted the external device must place an eight bit vector on the least significant byte of the data bus. This data is latched in S7.

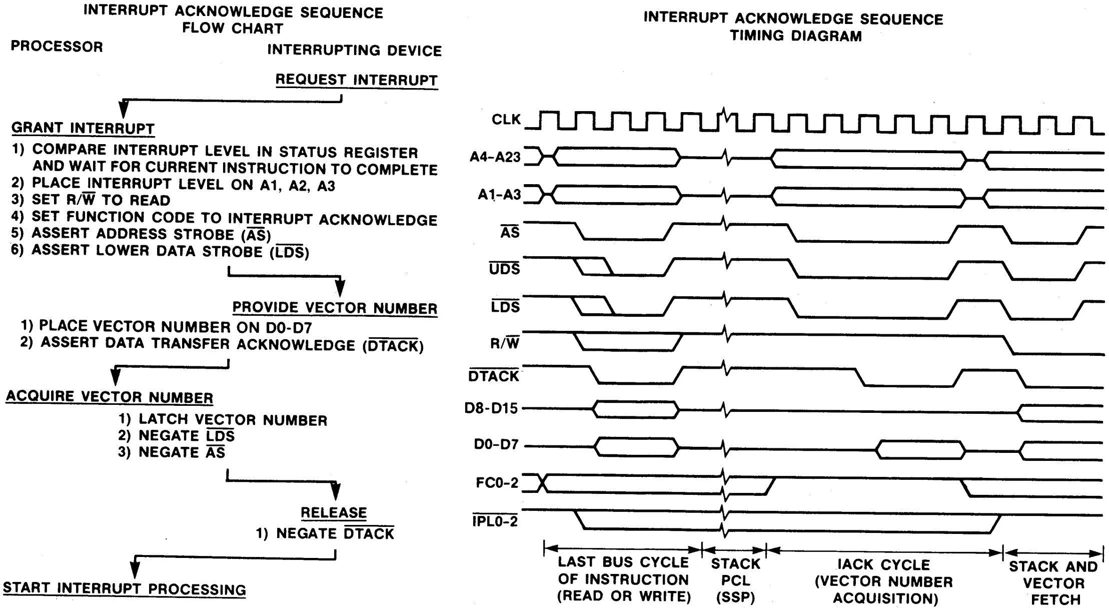

INTERRUPT ACKNOWLEDGE SEQUENCE
FLOW CHART
PROCESSOR
INTERRUPTING DEVICE
REQUEST INTERRUPT
GRANT INTERRUPT
1) COMPARE INTERRUPT LEVEL IN STATUS REGISTER AND WAIT FOR CURRENT INSTRUCTION TO COMPLETE
2) PLACE INTERRUPT LEVEL ON A1, A2, A3
3) SET R/W̄ TO READ
4) SET FUNCTION CODE TO INTERRUPT ACKNOWLEDGE
5) ASSERT ADDRESS STROBE (AS̄)
6) ASSERT LOWER DATA STROBE (LDS̄)
PROVIDE VECTOR NUMBER
1) PLACE VECTOR NUMBER ON D0-D7
2) ASSERT DATA TRANSFER ACKNOWLEDGE (DTACK̄)
ACQUIRE VECTOR NUMBER
1) LATCH VECTOR NUMBER
2) NEGATE LDS̄
3) NEGATE AS̄
RELEASE
1) NEGATE DTACK̄
START INTERRUPT PROCESSING
INTERRUPT ACKNOWLEDGE SEQUENCE
TIMING DIAGRAM
CLK
A4-A23
A1-A3
AS
UDS
LDS
R/W
DTACK
D8-D15
D0-D7
FC0-2
IPL0-2
LAST BUS CYCLE OF INSTRUCTION (READ OR WRITE)
STACK PCL (SSP)
IACK CYCLE (VECTOR NUMBER ACQUISITION)
STACK AND VECTOR FETCH

Figure 2

As shown on Table 1 vectors 64-255 are assigned to the user interrupt vector numbers. These vectors must be generated externally and placed on the data bus during an interrupt acknowledge cycle. In our interrupt controller we arbitrarily choose vectors 249-255 as the vectors assigned to the interrupting peripheral devices. Table 3a shows this assignment and how the vector can be decoded from the address bits A1-A3.

Priority level	Vector assigned	Data	Address
Device #	Decimal #	D7 D6 D5 D4 D3 D2 D1 D0	A3 A2 A1
1 (low priority)	249	1 1 1 1 1 0 0 1	0 0 1
2	250	1 1 1 1 1 0 1 0	0 1 0
3	251	1 1 1 1 1 0 1 1	0 1 1
4	252	1 1 1 1 1 1 0 0	1 0 0
5	253	1 1 1 1 1 1 0 1	1 0 1
6	254	1 1 1 1 1 1 1 0	1 1 0
7 (high priority)	255	1 1 1 1 1 1 1 1	1 1 1

TABLE 3a

The two PAL units used on this design example generate the interrupt vectors and all the necessary control signals. The various signals and their implementation on the PAL units are explained below.

INT7-INT0: Any of the seven peripheral devices can request an interrupt by asserting one of these inputs. The interrupt must remain asserted until acknowledged by the CPU.

FC2-FC0 and AS: Function code or processor status code become logical high during an interrupt acknowledge cycle. Address strobe is asserted anytime valid address is on the bus. DTACK and Data output control are decoded from these four outputs of the 68000.

A1-A3: The three least significant bits of the address bus contain the encoded level of the interrupting device. These signals are used in generating the vector number which is to be put on the data bus.

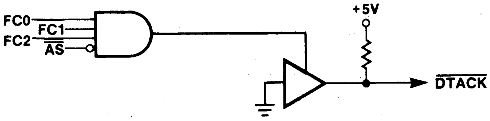
RESET: Reset is an input which clears all outputs, and is used for system initialization.

IPL2-IPL0: These PAL outputs are active low synchronous signals which interface directly to the CPU. They represent the encoded level of the highest priority interrupt that is requested. Table 3b shows the truth table of our priority encoder implemented on a PAL.

Interrupt request input from peripheral							Encoded int. level		
INT7	INT6	INT5	INT4	INT3	INT2	INT1	IPL2	IPL1	IPL0
0	X	X	X	X	X	X	0	0	0
1	0	X	X	X	X	X	0	0	1
1	1	0	X	X	X	X	0	1	0
1	1	1	0	X	X	X	0	1	1
1	1	1	1	0	X	X	1	0	0
1	1	1	1	1	0	X	1	0	1
1	1	1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1	1	1

TABLE 3b

DTACK: Data transfer acknowledge must be asserted by outside circuitry during a data transfer operation. The logic diagram shown below illustrates how /DTACK is derived from address strobe and processor status signals.



D7-D0: The three least significant bits of the data output can be decoded straight from the least significant address bits A3 A2 A1. That is D3=A3 D2=A2 D1=A1. The other five bits of data can be held high with pull up resistors. Output of the three data bits become enabled by using the same scheme which was used in enabling the DTACK output.

INTACK7-INTACK1: Only one of these signals will be asserted during the interrupt acknowledge cycle. This signal feeds back into the interrupting device to tell that device that its interrupt has been acknowledged. We can use the 3-bit addresses to decode these signals as shown in Table 3a.

			INTACK						
A3	A2	A1	7	6	5	4	3	2	1
0	0	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	0
0	1	0	1	1	1	1	1	0	1
0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	1	0	1	1	1
1	0	1	1	1	0	1	1	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1

TABLE 3c

The logic diagram of the controller is shown in Figure 3. The controller can operate without any wait states at the fastest CPU clock rate of 12.5 MHZ as shown in the timing diagram of Fig. 4. The appendix contains the fuse plot which translates into a logic schematic for these two PALs.

Auto-vectored Interrupts (Method 2)

The seven auto-vector interrupts are assigned vector numbers 25 through 31. Interrupts are requested by placing the encoded level of the request on the interrupt control pins (IPL2-IPL0). The processor responds to this request by placing the requested level on the processor's status register and by inhibiting all requests of lower priority. When the pending instruction cycle is completed an interrupt acknowledge cycle takes place. If Valid Peripheral Address, (VPA) is asserted before the falling edge of S4 then an auto-vector routine takes place. The data or the vector number is generated internally depending on the priority level of the interrupt request; the vector assigned is shown on the table at the beginning of this report. The Auto-vector timing diagram and a very simple and practical interrupt controller implemented on a PAL is shown in Fig. 4. The PAL DESIGN SPECIFICATIONS are included in the appendix. Note that VPA is generated by enabling the PAL output when FC2-FC1 are high and AS is asserted. The appendix contains the fuse plots for this PAL design. Note that the fuse plots translates directly to a logic diagram.

Interrupt Controller Logic Diagram
External Vector Generation

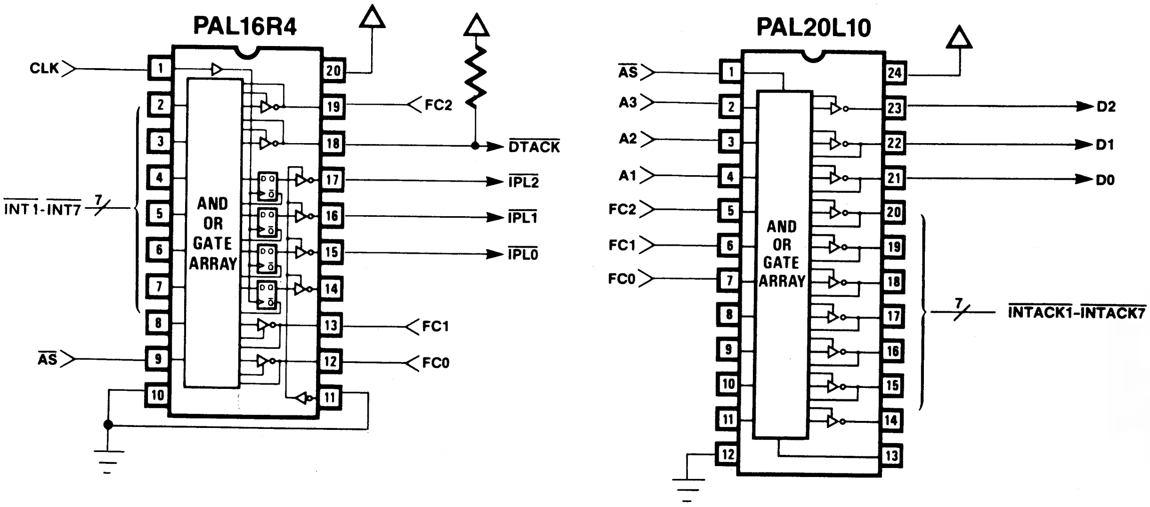
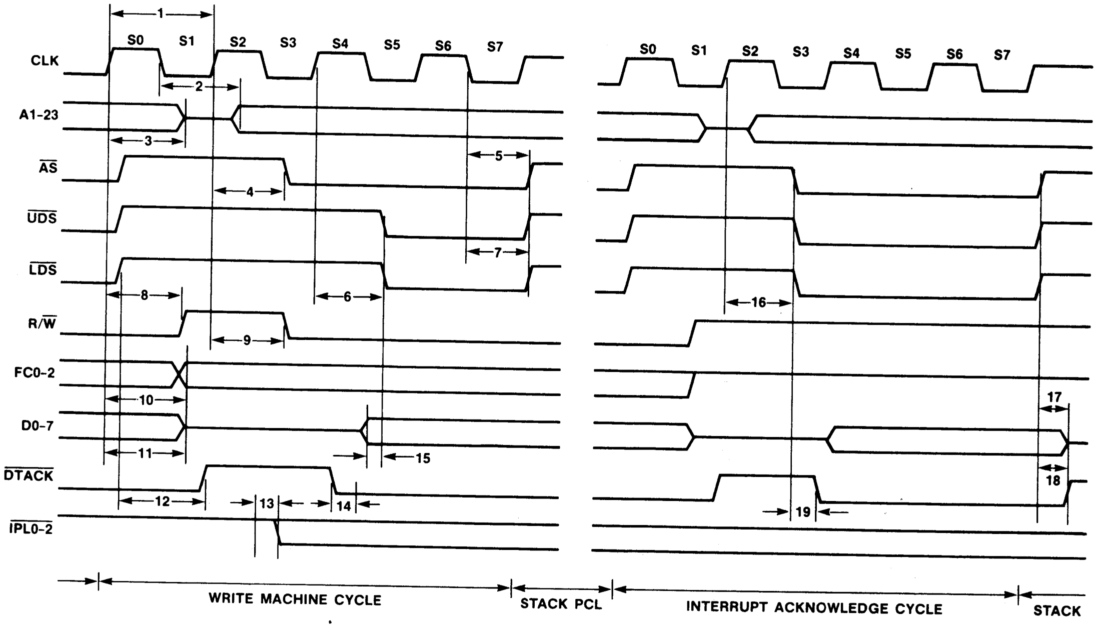


Figure 3



1 CLOCK PERIOD (80 ns)
2 CLOCK LOW TO ADDRESS (55 ns)
3 CLOCK HIGH TO ADDRESS Hi-Z (60 ns)
4 CLOCK HIGH TO $\overline{AS}$ LOW (55 ns)
5 CLOCK LOW TO $\overline{AS}$ HIGH (50 ns)
6 CLOCK HIGH TO $\overline{DS}$ LOW (55 ns)
7 CLOCK LOW TO $\overline{DS}$ HIGH (50 ns)
8 CLOCK HIGH TO R/$\overline{W}$ HIGH (60 ns)
9 CLOCK HIGH TO R/$\overline{W}$ LOW (60 ns)
10 CLOCK HIGH TO FC VALID (55 ns)
11 CLOCK HIGH TO DATA Hi-Z (60 ns)
12 $\overline{AS}$ HIGH TO $\overline{DTACK}$ HIGH (70 ns)
13 PAL's CLOCK TO OUT (25 ns)
14 MINIMUM SETUP TIME (20 ns)
15 DATA VALID TO $\overline{DS}$ LOW (15 ns)
16 CLOCK HIGH TO $\overline{AS}$, $\overline{DS}$ LOW (55 ns)
17 PAL's INPUT TO Hi-Z (25 ns)
18 PAL's INPUT TO Hi-Z (25 ns)
19 PAL's INPUT TO Hi-Z (25 ns)

Figure 4

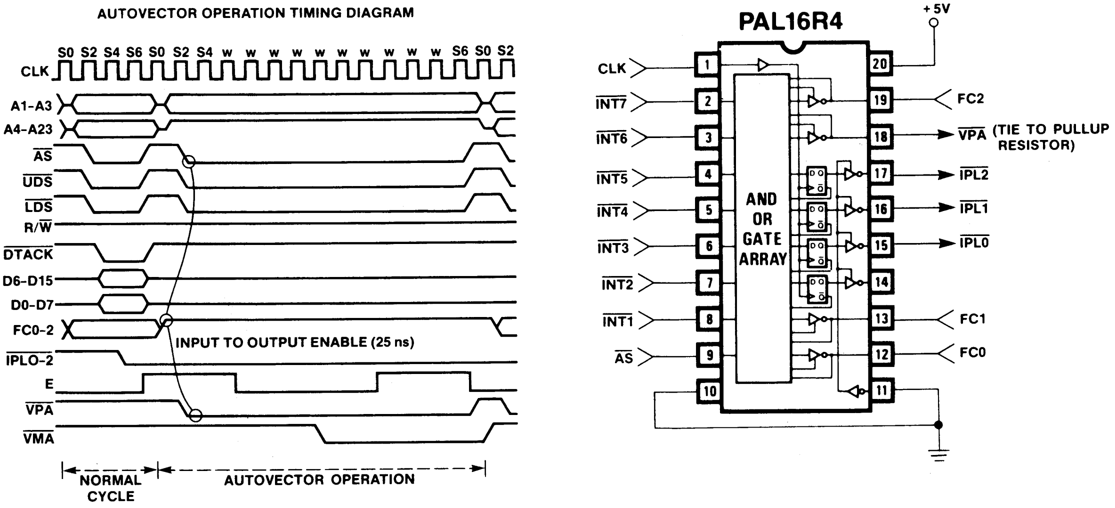


Figure 5

Conclusions

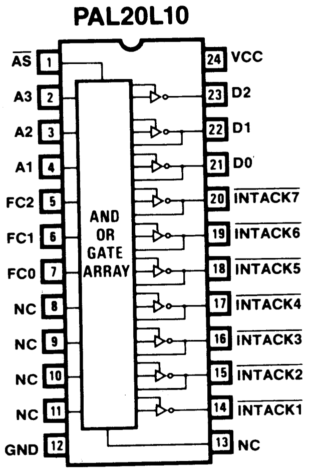
We have just seen how to implement an interrupt controller circuit using one or two PAL devices. This circuit can operate at the maximum operating frequency of the 68000 which is 12.5 MAZ. Since the most critical timing of the circuit is the PAL units's input to Hi-Z we see that we can operate the circuit with a wide spectrum of frequencies and with slower PAL units. To guarantee operation, timing spec. #16 and #19 on Figure 4 must add up to assert DTACK 20ns prior to falling edge of S4. Thus 55ns + (PAL unit's input to Hi-Z) should be less than or equal to 100ns for a 12.5 MHZ clock. We see that this qualifies fast, regular and half power PAL devices for this circuit. At slower CPU clock rates even the quarter-power PAL devices may be used.

References

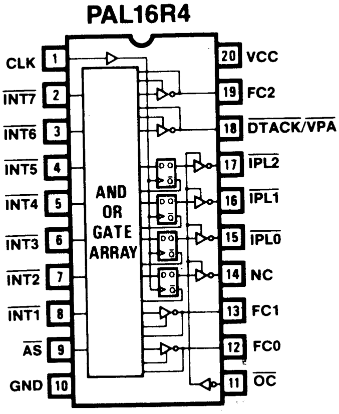
1. J. Birkner, V. Coli, "Programmable Array Logic Handbook," Monolithic Memories Inc.
2. MC68000 Data Sheet, Motorola Semiconductors. Austin, Texas 78721
3. Rex Davis, "Prioritized Individually Vectored Interrupts for Multiple Peripheral Systems with the MC68000," Motorola Application Note AN-819

Appendix

Interrupt Controller



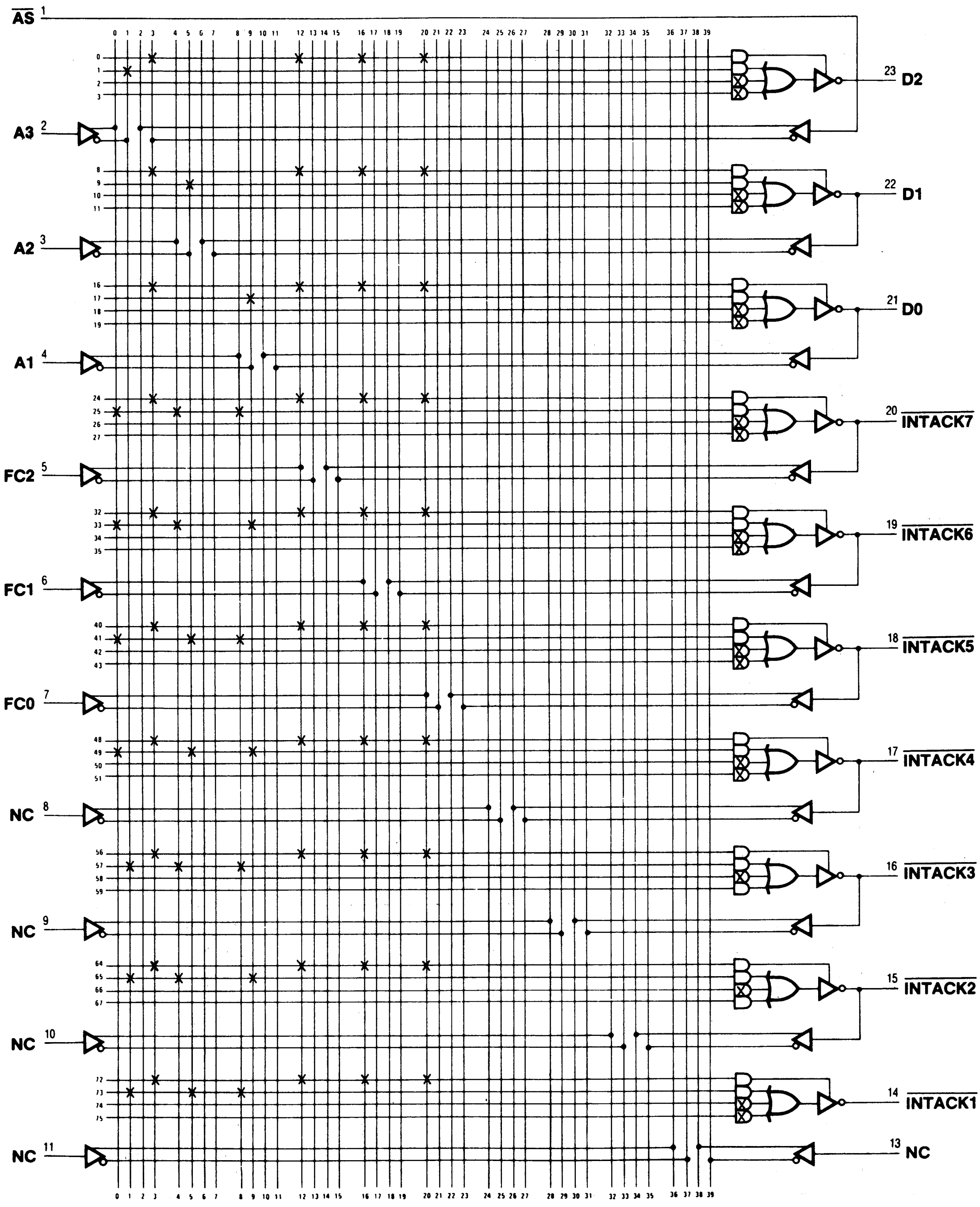
Interrupt Controller



```
PAL20L10                                                  PAL DESIGN SPECIFICATIONS
INTC.DAT                                                  DANESH TAVANA     9/25/82
INTERRUPT CONTROLLER
MMI, SUNNYVALE
/AS      A3       A2       A1       FC2      FC1      FC0      NC       NC  NC  NC  GND
 NC /INTACK1 /INTACK2 /INTACK3 /INTACK4 /INTACK5 /INTACK6 /INTACK7 D0  D1  D2  VCC


IF (FC2* FC1* FC0* AS) /D2 =/A3

IF (FC2* FC1* FC0* AS) /D1 =/A2

IF (FC2* FC1* FC0* AS) /D0 =/A1

IF (FC2* FC1* FC0* AS) INTACK7 = A3* A2* A1

IF (FC2* FC1* FC0* AS) INTACK6 = A3* A2*/A1

IF (FC2* FC1* FC0* AS) INTACK5 = A3*/A2* A1

IF (FC2* FC1* FC0* AS) INTACK4 = A3*/A2*/A1

IF (FC2* FC1* FC0* AS) INTACK3 =/A3* A2* A1

IF (FC2* FC1* FC0* AS) INTACK2 =/A3* A2*/A1

IF (FC2* FC1* FC0* AS) INTACK1 =/A3*/A2* A1



FUNCTION TABLE
/AS FC2 FC1 FC0 A3 A2 A1 D2 D1 D0
/INTACK7 /INTACK6 /INTACK5 /INTACK4 /INTACK3 /INTACK2 /INTACK1
;                                        /  /  /  /  /  /  /
;                                        I  I  I  I  I  I  I
;                                        N  N  N  N  N  N  N
;                                        T  T  T  T  T  T  T
;                                        A  A  A  A  A  A  A
;/  F  F  F                              C  C  C  C  C  C  C
;A  C  C  C    A  A  A    D  D  D        K  K  K  K  K  K  K
;S  2  1  0    3  2  1    2  1  0        7  6  5  4  3  2  1       COMMENTS
-------------------------------------------------------------------------------
 L  H  H  H    L  L  L    L  L  L        H  H  H  H  H  H  H       NO INTERRUPT
 L  H  H  H    L  L  H    L  L  H        H  H  H  H  H  H  L       LEVEL 1
 L  H  H  H    L  H  L    L  H  L        H  H  H  H  H  L  H       LEVEL 2
 L  H  H  H    L  H  H    L  H  H        H  H  H  H  L  H  H       LEVEL 3
 L  H  H  H    H  L  L    H  L  L        H  H  H  L  H  H  H       LEVEL 4
 L  H  H  H    H  L  H    H  L  H        H  H  L  H  H  H  H       LEVEL 5
 L  H  H  H    H  H  L    H  H  L        H  L  H  H  H  H  H       LEVEL 6
 L  H  H  H    H  H  H    H  H  H        L  H  H  H  H  H  H       LEVEL 7
 H  X  X  X    X  X  X    Z  Z  Z        Z  Z  Z  Z  Z  Z  Z       OUTPUT HI-Z
 L  H  H  L    X  X  X    Z  Z  Z        Z  Z  Z  Z  Z  Z  Z       OUTPUT HI-Z
 L  H  L  H    X  X  X    Z  Z  Z        Z  Z  Z  Z  Z  Z  Z       OUTPUT HI-Z
 L  L  H  H    X  X  X    Z  Z  Z        Z  Z  Z  Z  Z  Z  Z       OUTPUT HI-Z
-------------------------------------------------------------------------------
```

PAL1 — Interrupt Controller

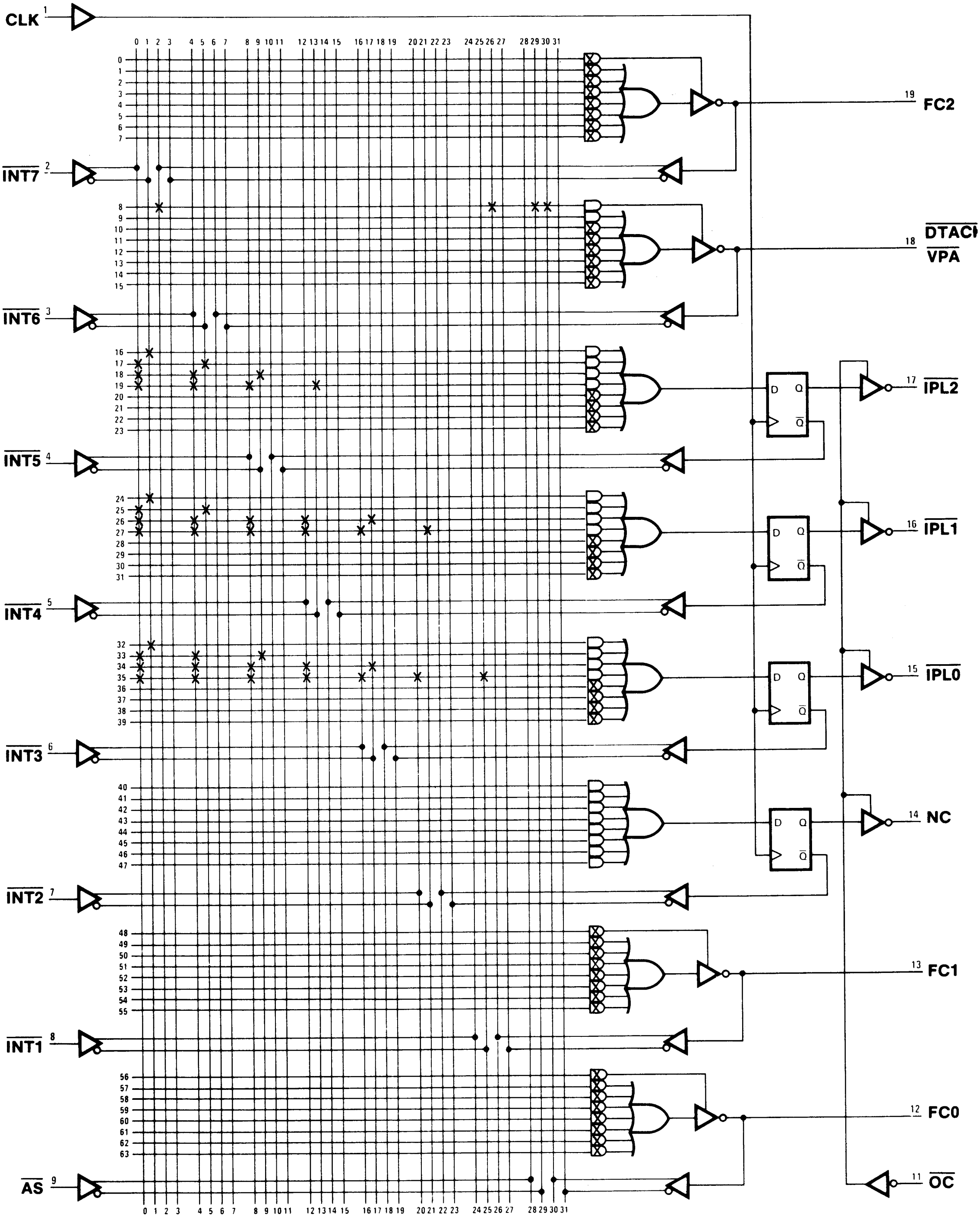
Logic Diagram PAL20L10



```
PAL16R4                                           PAL DESIGN SPECIFICATIONS
INTA.DAT                                          DANESH TAVANA
INTERRUPT CONTROLLER
MMI SUNNYVALE, CA
CLK /INT7 /INT6 /INT5 /INT4 /INT3 /INT2 /INT1  /AS GND
/OC  FC0    FC1    NC  /IPL0 /IPL1 /IPL2 /DTACK FC2 VCC

IF (FC2* FC1* FC0* AS) DTACK = VCC                     ;ASSERT IF OUTPUT ENABLE

IPL2 :=  INT7                                          ;PRIORITY ENCODED BIT
      + /INT7* INT6
      + /INT7*/INT6* INT5
      + /INT7*/INT6*/INT5* INT4

IPL1 :=  INT7                                          ;PRIORITY ENCODED BIT
      + /INT7* INT6
      + /INT7*/INT6*/INT5*/INT4* INT3
      + /INT7*/INT6*/INT5*/INT4*/INT3* INT2

IPL0 :=  INT7                                          ;PRIORITY ENCODED BIT
      + /INT7*/INT6* INT5
      + /INT7*/INT6*/INT5*/INT4* INT3
      + /INT7*/INT6*/INT5*/INT4*/INT3*/INT2* INT1




FUNCTION TABLE

CLK /INT7 /INT6 /INT5 /INT4 /INT3 /INT2 /INT1
FC2 FC1 FC0 /AS /IPL2 /IPL1 /IPL0 /DTACK
;                                    /
;   / / / / / / /          / / /  D
;   I I I I I I I          I I I  T
;C  N N N N N N N  F F F / P P P  A
;L  T T T T T T T  C C C A L L L  C
;K  7 6 5 4 3 2 1  2 1 0 S 2 1 0  K           COMMENTS
-------------------------------------------------------------------------
 C  L X X X X X X  H H H L  L L L  L          ASSERT /DTACK
 C  H L X X X X X  H H H H  L L H  Z          INTERRUPT LEVEL 7
 C  H H L X X X X  L L H L  L H L  Z          INTERRUPT LEVEL 6
 C  H H H L X X X  L H L L  L H H  Z          INTERRUPT LEVEL 5
 C  H H H H L X X  L H H L  H L L  Z          INTERRUPT LEVEL 4
 C  H H H H H L X  H L L L  H L H  Z          INTERRUPT LEVEL 3
 C  H H H H H H L  H L H L  H H L  Z          INTERRUPT LEVEL 2
 C  H H H H H H H  H H L L  H H H  Z          INTERRUPT LEVEL 1
-------------------------------------------------------------------------
```

PAL2 — Interrupt Controller

Logic Diagram PAL16R4



Registered PROMs Impact Computer Architecture

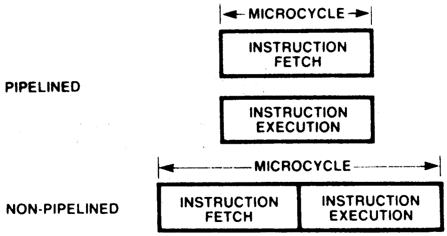
John Birkner

A family of registered PROMs offers new savings for designers of pipelined microprogrammable systems. The wide instruction register, which holds the microinstruction during execution, is now incorporated into the PROM chip. This feature saves power, improves cycle times and decreases printed circuit board area over the present technique of using an external instruction register. Those designs which were previously non-pipelined can now be upgraded with little additional cost.

Registered PROMs Impact Computer Architecture by John Birkner

Pipelined Microprogrammable Systems

Microprogrammed processors and controllers can generally be classified as either pipelined or non-pipelined. The difference is demonstrated by the microcycle timing diagrams below:



Clearly, the benefit of pipelining is that the microcycle time is defined by the **longer of either** fetch or execution times, rather than the **sum of both** fetch and execution times. This gain can be as much as 2:1, if fetch and execution times are equal. The gain in cycle time is, of course, lost when a branch requires the look ahead fetch to be discarded. The ratio of sequential fetches to branches varies according to application. In heavily decision-oriented applications, the ratio may be as low as 3:1. Typically, the ratio is 5:1. An example of a pipelined microprogrammed processor is shown in Figure 1.

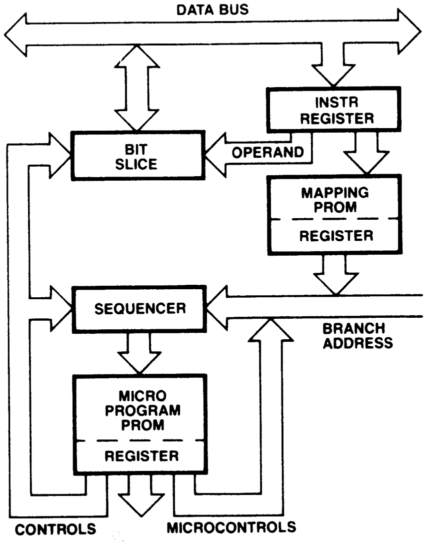


Figure 1. Pipelined Microprogrammed Processor

Benefits

Advantages of registered PROMs in pipelined microprogrammable systems are:

- **Lower package counts**
- **Lower power consumption**
- **Faster cycle times**

In a benchmark system requiring a control store of 64 bits by 1024 or 2048 words, these benefits are equated to:

- **8 external register package savings**
- **Over 1 amp max I_{CC} reduction (compared to Schottky registers)**
- **280mA max I_{CC} reduction (compared to low power Schottky registers)**
- **20nsec faster cycle time (compared to low power Schottky registers)**

Structured Logic

The registered PROM is a structured logic primitive which, along with other primitives, can be used as building blocks to define a variety of processor and controller architectures.

The most basic architecture is formed by feeding the outputs of a registered PROM back to the address inputs.

This state machine can sequence from one step to the next as a function of present state and test inputs.

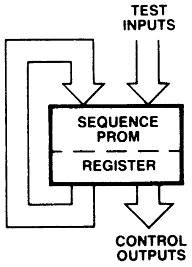


Figure 2. State Sequencer

A powerful microprogram sequencer can be constructed from registered PROMs, as shown in Figure 3. The Next Address PROM provides the normal sequence while the Branch Address PROM provides instant availability of an alternate sequence when the branch condition is satisfied. The Return Jump RAM provides a microsubroutine capability.

An 8-bit microprogrammed computer can be constructed, Figure 4, using as little as 30 ICs. A 100nsec microcycle is easily achievable. Instruction decode is accomplished by direct vector from 8-bit macro instruction into microspace. Dynamic RAM control signals RAS, CAS and WRITE are under direct microprogram control.

The 8-bit computer, Figure 4, is a particularly good example of how the registered PROM takes its place among RAMs, registers and buffers as a basic building block in high performance microprogrammable systems.

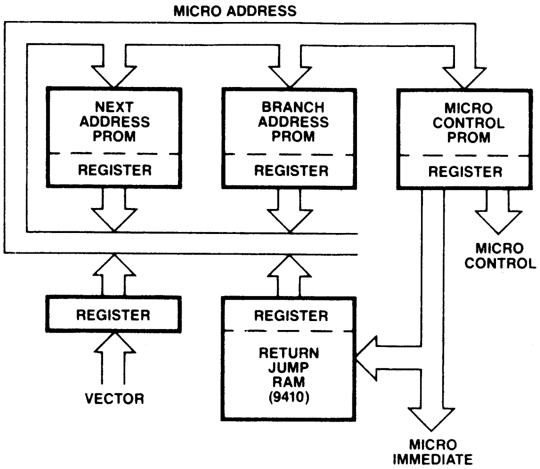


Figure 3. Microprogram Sequencer

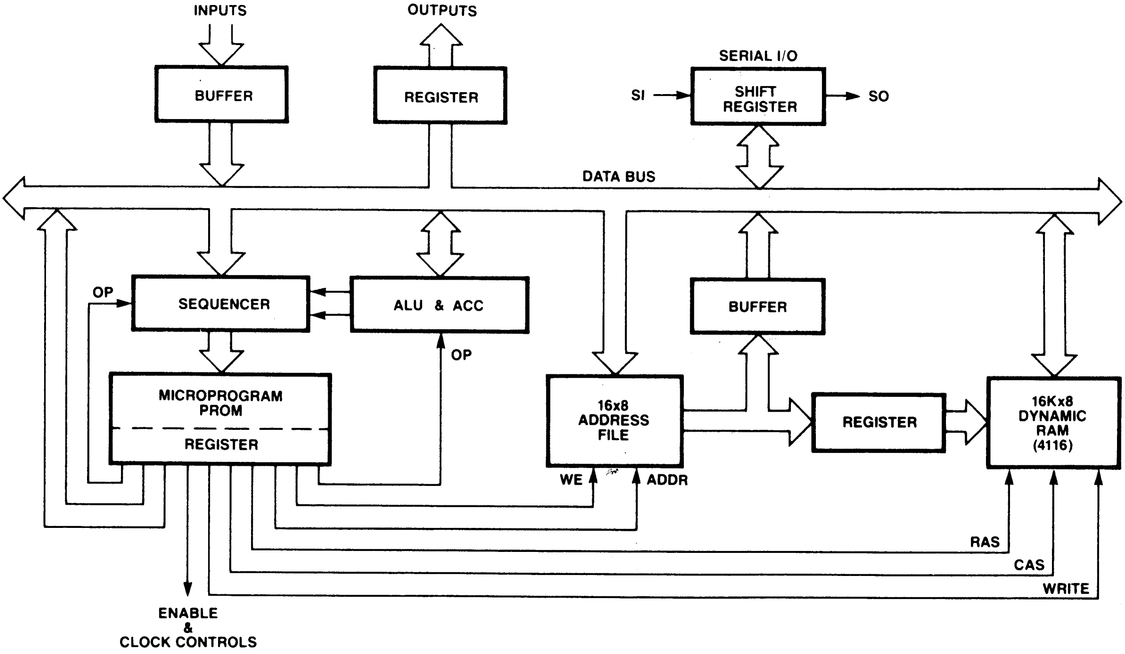


Figure 4. 8-Bit Computer

PROMs and PLEs: An Application Perspective

By Zahir Ebrahim

Abstract

Programmable Read Only Memories are widely used in digital systems, both as a memory device, as well as a Programmable Logic Element (PLE™). This document describes the use of PROMs and PLEs in many practical applications ranging from Diagnostic Microsequencers, to M-Bit Parallel CRC. The concept of testability and built-in Diagnostics in the PROMs is also illustrated.

PROMs and PLEs: An Application Perspective

Zahir Ebrahim

PROM applications can be divided into two categories:

- **Programmable Read Only Memory**
- **Programmable Logic Element (PLE)**

Memory Device

The PROM as a memory device has become an invaluable storage element in digital systems. It is used in data and control paths of a system to store constant data and firmware microprograms respectively, as shown in Figure 1.

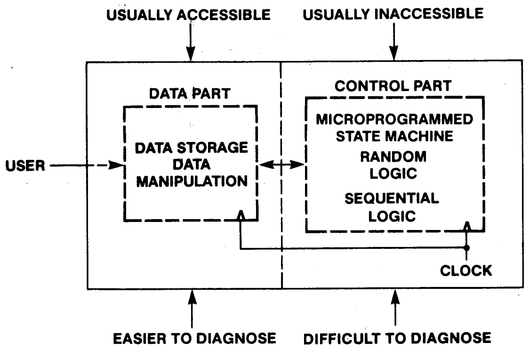


Figure 1. A Typical Digital system, viewed as an interaction of Data and Control parts.

In the data path, a PROM is primarily used for table look-up applications. Trigonometric functions, signal processing coefficients, target identification and terrain data for military applications, software programs for bootstrap and initialization etc., are all usually stored in a PROM.

In the control path, a PROM is mainly used to store microprograms. Microprogramming has become increasingly popular as a method of sequencing and control of states in a digital system (Figure 2). Microprogrammed controllers may be as simple as a standalone PROM-Register finite state machine (Figure 3). Or they may be quite complex like a microprogrammed CPU, in which the PROM stores the instruction set of the system (Figure 8). These microprogrammed controllers are widely used in computers, disk controllers, smart peripherals, video games, industrial control systems etc. Appendix II identifies some areas of applications for PROMs. Appendix III compares PROMs with EPROMs.

Registered PROMs

Most memory applications of PROMs involve the PROM data to be stored in a temporary register before it is used (Pipelining as in Figure 2). Therefore, Monolithic Memories' new generation of PROMs also contain a D-type edge-triggered register on board. These Registered PROMs are generally faster in operation than the discrete PROM-Register combination, and also occupy less board space.

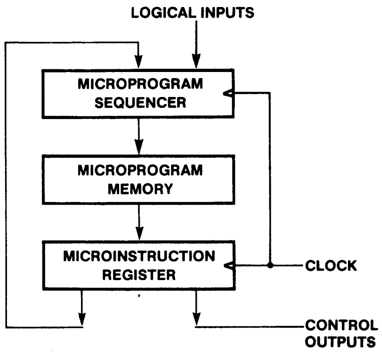


Figure 2. Typical mircroprogrammed sequencer. The next state depends on the present state and present logical inputs.

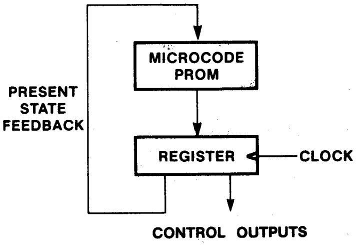


Figure 3. PROM-Register microsequencer. The next state depends only on the present state.

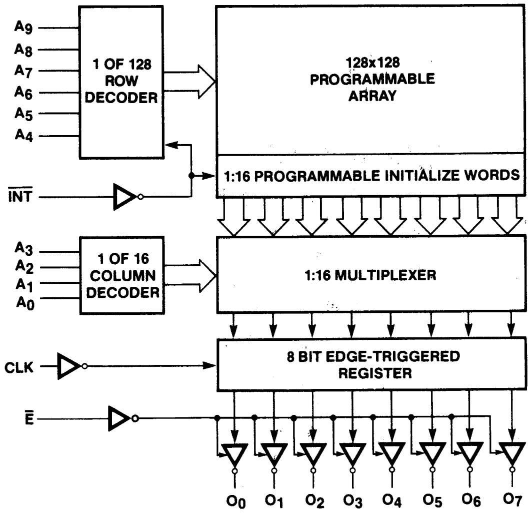


Figure 4. Block diagram of 24 pin Registered PROM 63RA1681. The 2K x 8 PROM contains an additional 16 programmable Initialization words. Access time is 50 nsec.

Diagnostic PROMs

The Testing Problem

With the increasing complexity of digital systems and denser integration of devices, testing and diagnosing a fault in a system has become increasingly expensive. Internal signals, whether inside a chip or on a board, are usually not accessible from the outside world, as shown in Figure 5a. This invisibility of internal nodes makes it difficult to control and observe the internal states without extra logic circuitry, complex multiplexing schemes or a large number of extra pins. Space being at a premium, both in a chip and on the board, these schemes are seldom used.

Design for testability techniques like Level Sensitive Scan Design (LSSD), and external Serial Scan Diagnostics (SSD) are quite elegant in concept, but do not guarantee system integrity during diagnostics. Both techniques use the system registers for shifting diagnostic data as shown in Figure 5b. Since the same registers are used for normal operation as well as diagnostic function, they may cause a hazardous condition if a register selectively enables data onto a bus as shown in Figure 5b. An illegal combination of ones and zeros during the shifting process may enable more than one device on the bus, causing bus conflict and possible damage to the system.

Because of these limitations, designers have been reluctant to incorporate testability in the design phase, relying more and more on the ability of the test engineer to generate elaborate test vectors, or the expertise of the field engineer to diagnose and debug their systems.

A new PROM architecture has been developed by Monolithic Memories to solve this testing problem. It employs the concept of Serial Scan Diagnostic, but separates the diagnostic path from the system data path as shown in Figure 5c.

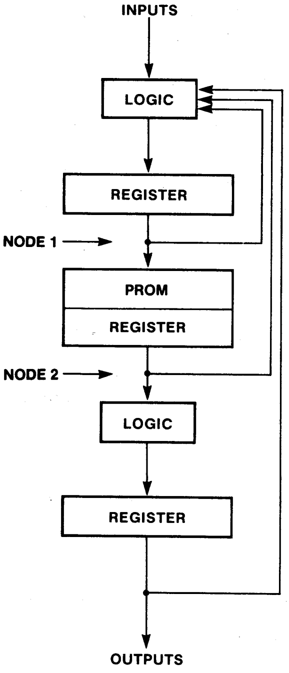


Figure 5a. Sequential testing nightmare. Internal nodes 1 and 2 are not directly visible or controllable.

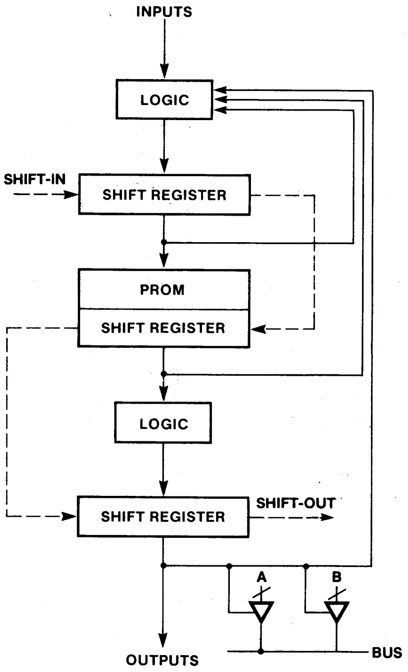


Figure 5b. Using the system registers for Serial Scan Diagnostic or LSSD may cause illegal states in the registers while shifting. Two or more devices might be accidently enabled on the same bus.

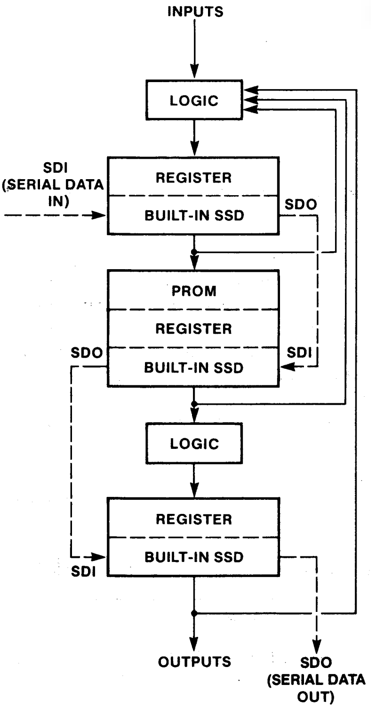


Figure 5c. Built-in Serial Scan Diagnostics eliminate shifting hazards. Separate Diagnostic and System data paths make the diagnostic shifting process invisible to the rest of the system.

The Diagnostic PROM contains a buried shift register (shadow register) in addition to an output register as shown in Figure 6.

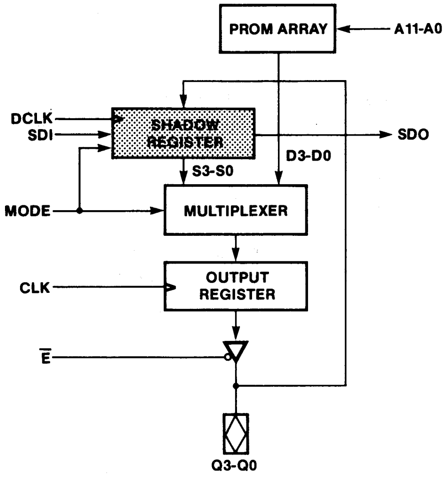


Figure 6. Block diagram of 4Kx4 Diagnostic PROM 63D1641.

This architecture enables writing external diagnostic data into the PROM output register without the concomitant shifting hazards. The test data is serially shifted into the shadow register, and then loaded into the output register. The contents of the output register is similarly observed by first loading it into the shadow register, and then serially shifting it out. The Diagnostic PROM also allows observation of bus data by disabling the output register, and loading the contents of the bus into the shadow register.

Thus the shifting-in of test vectors and shifting-out of internal states is not visible to the system, except along dedicated diagnostic data paths.

Advantages of Built-in Diagnostics

- On-chip diagnostics in the PROM eliminate the need to store testing data which typically occupies more than 30% of the PROM, the rest being taken up by the actual microprograms. Real time software generated test vectors can be shifted-in and out of the system whenever diagnostics need to be performed, instead of storing them permanently in the PROM. This provides flexibility in testing strategies and makes it easier to modify or generate new diagnostic routines. The system can be tested independently at the chip, board, module and system levels without having to wait for the entire system to be built before it can be tested.
- Functional testing is also made easy with the diagnostic architecture. Programs written in a high-level language can be compiled into test vectors which can be shifted into the system. The response can be converted back into a higher level functional description and displayed to the user.
- The Diagnostic PROMs are ideally suited for fault-tolerant applications. The PROM outputs can be continually sampled via the shadow register for self-checking diagnostics. Response to test vectors can be shifted out for compaction, and signature analysis performed by a table look-up operation in another PROM, with only the discrepancies being reported to the user. This eliminates the deluge of information generated in thorough and exhaustive tests.
- Maintenance and operational checkup times are also significantly reduced in systems incorporating built-in diagnostics. Faults can be quickly isolated without cumbersome board testing, or bothering with complicated test equipment.
- With testability being recognised as more a responsibility of the design engineer, than the aftermath efforts at salvation by the test and field engineers, the diagnostic PROMs can alleviate the designer's burden by providing him at least 3-to-1 reduction in chip count, with full diagnostic functionality already built-in. It consumes 50% less power, has greater speed and reliability due to device integration, and improves the overall system performance immeasurably.

The diagnostic architecture is being incorporated in several of the next generation of devices including Diagnostic Registers and Diagnostic PALs.

Function Table 63D1641

INPUTS				OUTPUTS			OPERATION
MODE	SDI	CLK	DCLK	Q_3-Q_0	S_3-S_0	SDO	
L	X	↑	*	Qn ← PROM	HOLD	S_3	Load output register from PROM array
L	X	*	↑	HOLD	$S_n \leftarrow S_{n-1}$ $S_0 \leftarrow$ SDI	S_3	Shift shadow register data
L	X	↑	↑	Qn ← PROM	$S_n \leftarrow S_{n-1}$ $S_0 \leftarrow$ SDI	S_3	Load output register from PROM array while shifting shadow register data
H	X	↑	*	$Q_n \leftarrow S_n$	HOLD	SDI	Load output register from shadow register
H	L	*	↑	SDI	$S_n \leftarrow Q_n$	SDI	Load shadow register from output bus
H	H	*	↑	HOLD	HOLD	SDI	No operation

* Clock must be steady or falling.

Examples

Control Path Examples:

Diagnostic and Registered PROMs can be used to implement simple standalone microsequencers which are powerful enough to control disk and tape drives, smart peripherals etc. Figure 7 illustrates a PROM-Register microsequencer constructed from Diagnostic PROMs. The outputs of the PROM are completely controllable and observable, making the entire system in which it is used, testable and reliable. Figure 8 shows the diagnostic and system data flow paths in a CPU.

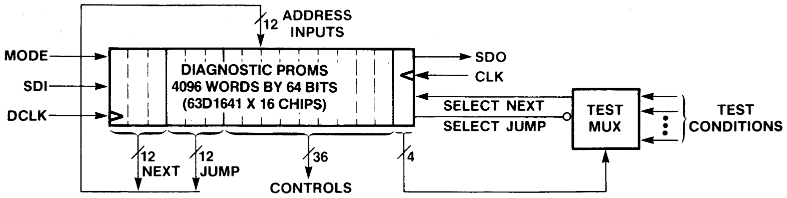


Figure 7. 64-Bit Microcontroller using sixteen 4K x 4 Diagnostic PROMs and one PAL. Operating speed is greater than 16 MHz.

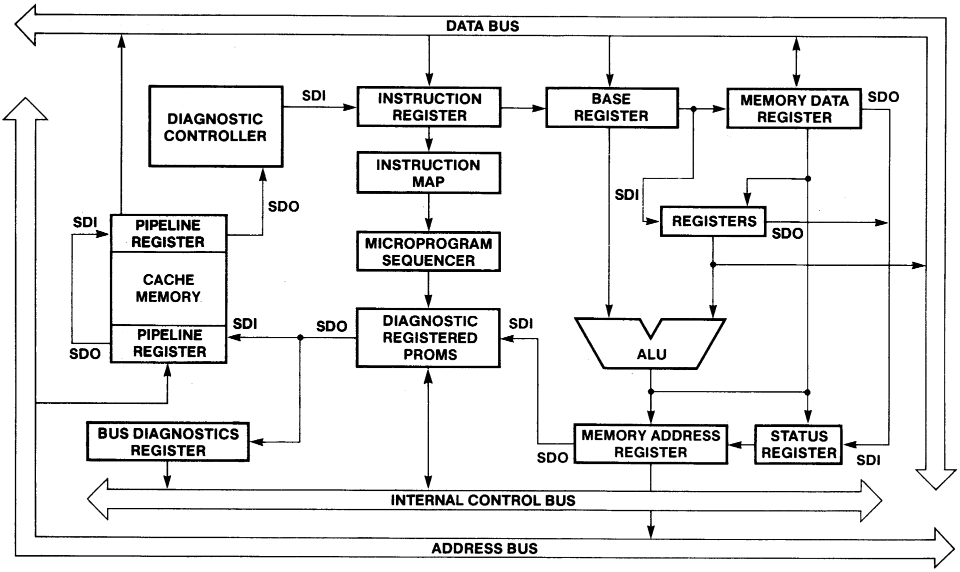


Figure 8. Block diagram of a pipelined CPU using Diagnostic PROMs and Diagnostic Registers. Internal states including the buses are controllable and observable.

Data Path Example:

A data path application of the PROM is in table look-up operations. In color signal processing for image restoration, enhancement and reproduction, homomorphic signal transformations are used to transform the image from one set of parameters to another. This transformation can generally be computed beforehand and stored in a PROM.

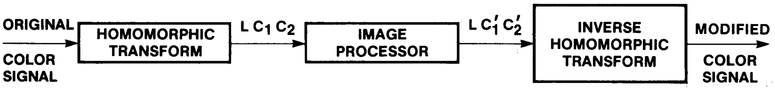


Figure 9. Block diagram of a general Color Signal Processing System.

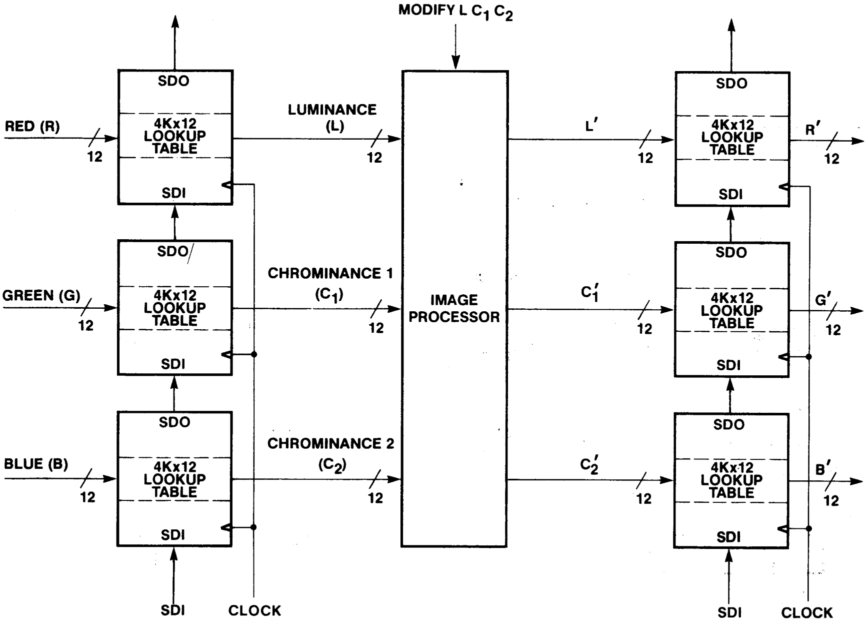


Figure 10. Eighteen 4K x 4 Diagnostic PROMs (63D1641) used in a typical Color Image Processing System. The serial Diagnostic capability allows control and observation of the processor.

Programmable Logic Element

The AND-OR planar structure of the PROM lends itself naturally to being viewed as a two-level logic circuit. The inputs to the PROM are fully decoded into all possible combinations in the fixed AND plane. Each combination (product term) is fuse connected to each output in the programmable OR plane.

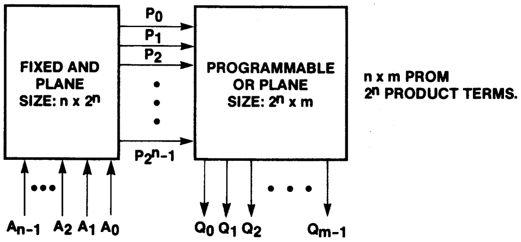


Figure 11. Block diagram of a PROM viewed as a PLE.

In terms of a PLE, a product term is equivalent to an AND gate equal in size to the number of inputs. Each output is equivalent to an OR gate connected to all the AND gates. Programming a fuse then implies breaking a connection between an AND gate and OR gate.

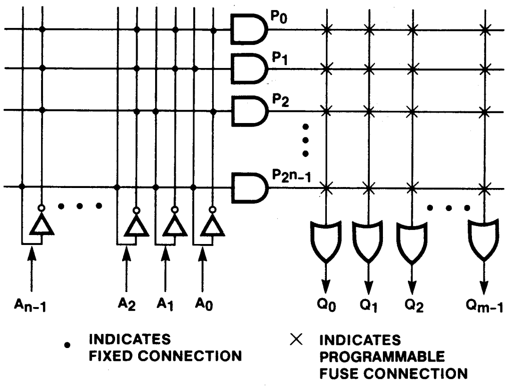

P0
P1
P2
P2n-1
An-1
A2
A1
A0
Q0
Q1
Q2
Qm-1
• INDICATES FIXED CONNECTION
× INDICATES PROGRAMMABLE FUSE CONNECTION

Figure 12. Equivalent Logic circuit. A virgin PLE has all the fuses intact.

Thus a PROM has a convenient structure for implementing combinatorial logic when a large number of input combinations are required or a large number of product terms per output is desired.

In terms of a Karnaugh map for a PLE, each minterm in the map corresponds to one product term in the PLE. Two or more adjacent minterms cannot be combined to generate a prime implicant, or eliminate a logic hazard. For example, the following Karnaugh map implemented in a PLE will generate the function $f = a\bar{b} + ab$. The minimized function $f = a$ as indicated by the dotted prime implicant can not be implemented. The PLE does not contain a product term with fewer than all its inputs present.

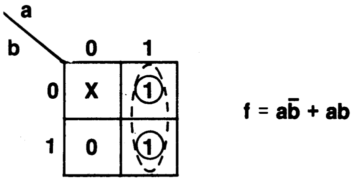


Figure 13. An arbitrary Karnaugh map for a PLE.

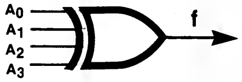
The absence of prime implicants in a PLE may cause logic hazards which may be unavoidable in asynchronous control systems. However these hazards are masked out in synchronous control systems by the registers, and are largely irrelevent in data path applications where only the final steady state results are looked at. Indeed, most applications of PLEs are in synchronous systems to replace random logic. In the data paths, they are used to generate complex functions like ALU operations, Pseudo Random Number sequences, Error Detection codes etc.

Appendix I reviews programmable logic devices, and compares PLAs, PALs and PLEs.

Examples

Control Path Application of PLE

A common application of PLEs in the control path is to replace random logic or customize logic functions. An n input Exclusive OR function is quite commonly required in comparator and adder circuits. It contains 2^{n-1} product terms, which increases exponentially with n. Therefore, it is very efficient to implement large XOR functions in a PROM. Figure 14 shows the implementation of a 4-input XOR in a PLE.



$$\begin{aligned} f = {} & A_0 \oplus A_1 \oplus A_2 \oplus A_3 \\ + {} & A_0 \overline{A_1}\,\overline{A_2}\,\overline{A_3} + \overline{A_0} A_1 \overline{A_2}\,\overline{A_3} \\ + {} & \overline{A_0}\,\overline{A_1} A_2 \overline{A_3} + \overline{A_0}\,\overline{A_1}\,\overline{A_2} A_3 \\ + {} & A_0 A_1 A_2 \overline{A_3} + A_0 A_1 \overline{A_2} A_3 \\ + {} & A_0 \overline{A_1} A_2 A_3 + \overline{A_0} A_1 A_2 A_3 \end{aligned}$$

A_2, A_3 \ A_0, A_1	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

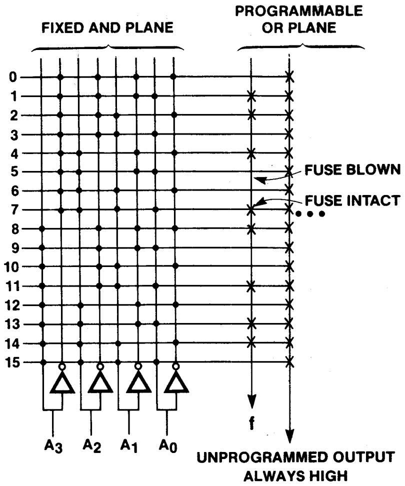


Figure 14. 4-input XOR function implemented in a PLE.

Data Path Application of PLE

In the data path, a PLE can be used to implement complex functions such as a Pseudo Random Number (PRN) Generator. PRN sequences are useful in encoding and decoding of information in signal processing and communication systems. They are used for data encryption in secure communication links, and error detection and correction codes in data communication systems. PRN sequences are also utilized as test vectors for circuits simulation, as signal modulators in radar rangefinding systems, and as reference white noise in many signal processing applications.

There are many techniques for generating PRN sequences. The most common technique is to use 'n' stages of linear shift registers with feedback through a logic function. The feedback logic function and the feedback paths determine a polynomial which characterises a PRN sequence. Figure 15 illustrates a typical mechanism for generating PRN sequences.

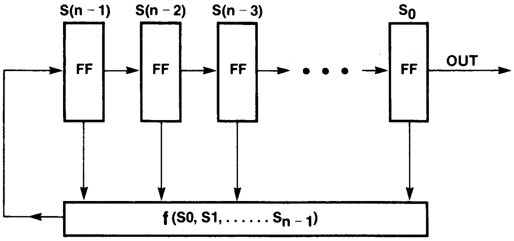


Figure 15. An 'n' Stage Linear Feedback Shift Register (LFSR). The PRN sequence generated is characterized by a polynomial of degree n. the feedback terms and logic function determine its binary coefficients.

The advantage of using a PLE for implementing PRN sequences is that any polynomial can be quickly customized in it. In data encryption systems where the code is frequently changed to protect from mischievous eavesdroppers, a PLE can be used to generate a new code each time, or several codes can be implemented in the same PLE.

An example of a PRN generator implemented in a Registered PROM is shown in Figure 16. A linear 2-input XNOR function is used to generate a PRN sequence characterized by a polynomial of degree 3. The PRN sequence is of maximum length with period 7.

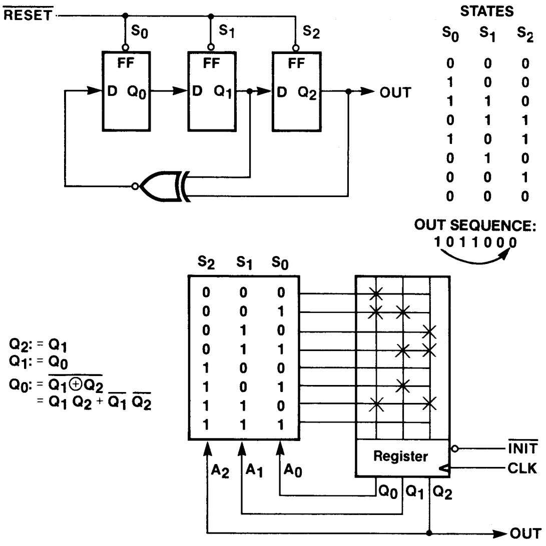


Figure 16. A 3-Stage Pseudo Random Number Generator implemented in a Registered PLE.

Parallel CRC in Registered PLEs

Cyclic Redundancy Check (CRC) is widely used for Error Detection in data communication. Both serial and parallel CRC can be performed depending on the nature of application. In serial data transfers on Local Area Networks, or between peripheral and main mamory, serial CRC is the preferred and perhaps the most efficient technique. However, systems employing wide data buses for high-speed short-distance data transfers, require a high-speed mechanism of ensuring data integrity. In these applications, parallel CRC might be the better alternative.

The implementation of an M-bit parallel CRC is much more complex than its serial counterpart. Although both use Linear Feedback Shift Register (LFSR) configurations, the parallel implementation requires an M-bit carry look-ahead circuitry to process the M data bits simultaneously (see reference 7). The equations for this carry look-ahead represent the output of each stage in the LFSR after every shift of an M-bit string of data. These equations contain a large number of XOR operations which make it very efficient to implement in a Registered PLE.

To illustrate with a practical example, Figure 18 shows the serial implementation of the CRC generator polynomial

$$G(X) = X^{16} + X^{12} + X^5 + 1$$

also called the CRC-CCITT standard. Figure 19 shows the 8-bit carry look-ahead equations for an 8-bit parallel CRC implementation of the same polynomial. These equations are derived in reference 7, where an implementation in 4 PAL devices is also shown with a maximum delay of 90 nsec. Figure 20 shows an implementation in only three 24-pin Registered PLEs and one SSI chip. The maximum delay is 50 nsec.

The speed of operation of parallel CRC implemented in Registered PLEs will remain the same for any generator polynomial and M. Increasing the complexity of the carry look-ahead equations only increases the number of devices required to implement them. It does not increase the delay.

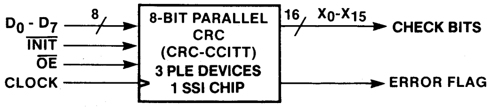


Figure 17. Block diagram of 8-bit parallel CRC.

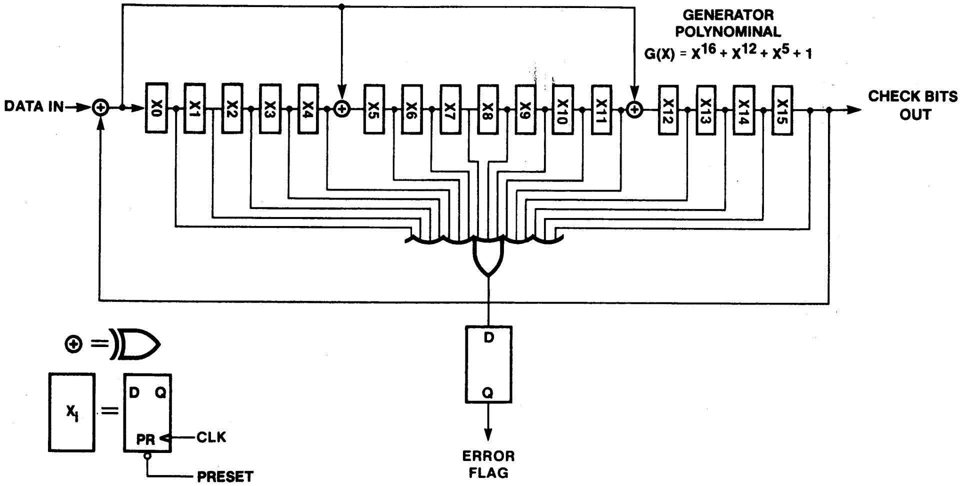
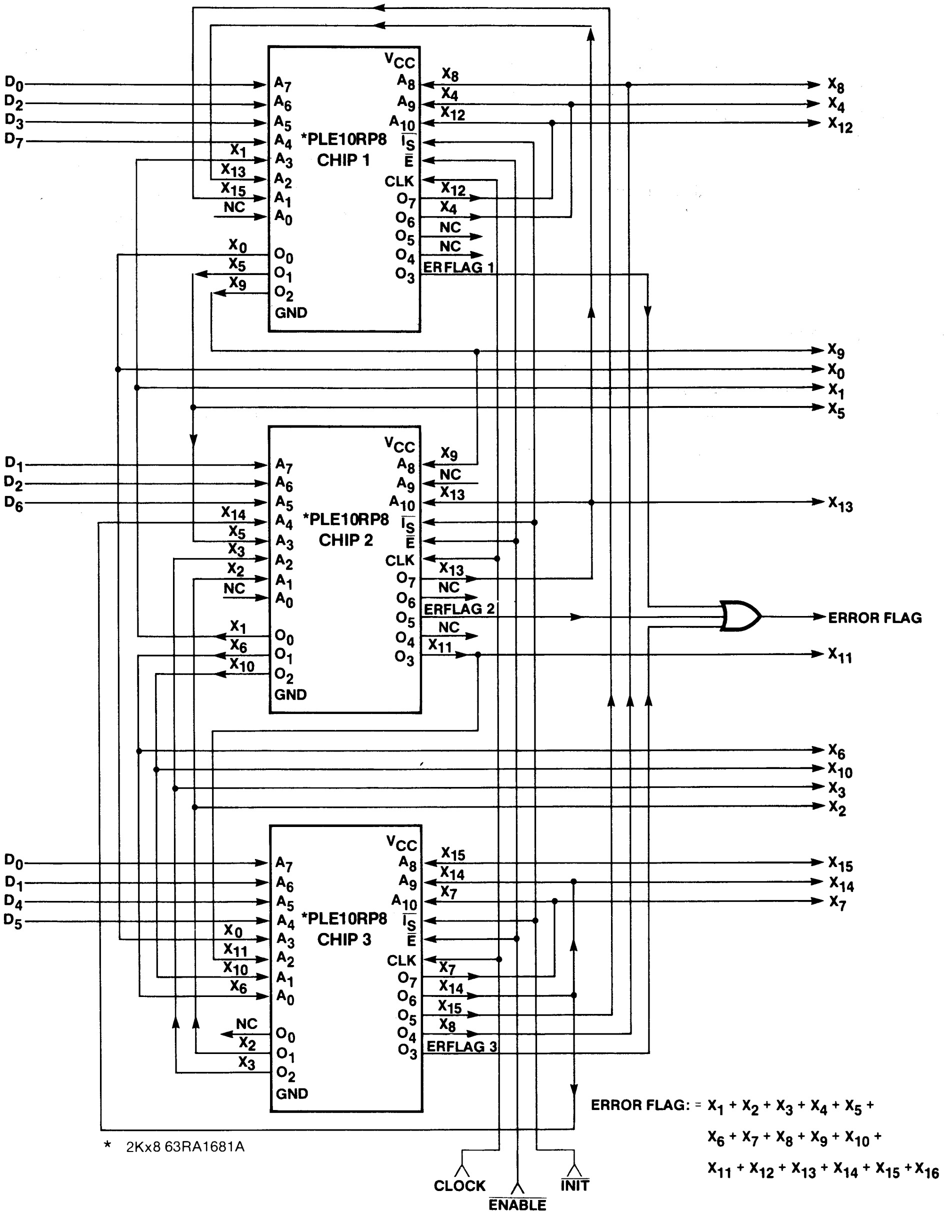


Figure 18. A 16-Bit Linear Feedback Shift Register (LFSR) implementing a Serial CRC Generator.

$X0\ (n+1) := X8\ (n) \oplus X12(n) \oplus D(3) \oplus D(7)$ chip1
$X1\ (n+1) := X9\ (n) \oplus X13(n) \oplus D(2) \oplus D(6)$ chip2
$X2\ (n+1) := X10(n) \oplus X14(n) \oplus D(1) \oplus D(5)$ chip3
$X3\ (n+1) := X11(n) \oplus X15(n) \oplus D(0) \oplus D(4)$ chip3
$X4\ (n+1) := X12(n) \oplus D3$ chip1
$X5\ (n+1) := X8\ (n) \oplus X12(n) \oplus X13(n) \oplus D(2) \oplus D(3) \oplus D(7)$ chip1
$X6\ (n+1) := X9\ (n) \oplus X13(n) \oplus X14(n) \oplus D(1) \oplus D(2) \oplus D(6)$ chip2
$X7\ (n+1) := X10(n) \oplus X14(n) \oplus X15(n) \oplus D(0) \oplus D(1) \oplus D(5)$ chip3
$X8\ (n+1) := X0\ (n) \oplus X11(n) \oplus X15(n) \oplus D(0) \oplus D(4)$ chip3
$X9\ (n+1) := X1\ (n) \oplus X12(n) \oplus D(3)$ chip1
$X10(n+1) := X2\ (n) \oplus X13(n) \oplus D(2)$ chip2
$X11(n+1) := X3\ (n) \oplus X14(n) \oplus D(1)$ chip2
$X12(n+1) := X4\ (n) \oplus X8\ (n) \oplus X12(n) \oplus X15(n) \oplus D(0) \oplus D(3) \oplus D(7)$ chip1
$X13(n+1) := X5\ (n) \oplus X9\ (n) \oplus X13(n) \oplus D(2) \oplus D(6)$ chip2
$X14(n+1) := X6\ (n) \oplus X10(n) \oplus X14(n) \oplus D(1) \oplus D(5)$ chip3
$X15(n+1) := X7\ (n) \oplus X11(n) \oplus X15(n) \oplus D(0) \oplus D(4)$ chip3

where Xi (n + 1) is the next state value of the corresponding
register i, i = 0, ..., 15
Xi (n) is the present value of the corresponding
register i, i = 0, ..., 15
D (n) is the parallel input data bits, where n = 0, ..., 7

Figure 19. Carry look-ahead equations for 8-bit parallel CRC with G (X). The equations are partitioned into 3 parts for efficient implementation in 3 chips.



ERROR FLAG: = $X_1 + X_2 + X_3 + X_4 + X_5 +$
$X_6 + X_7 + X_8 + X_9 + X_{10} +$
$X_{11} + X_{12} + X_{13} + X_{14} + X_{15} + X_{16}$

* 2Kx8 63RA1681A

Figure 20. Diagram showing how to connect 3 Registered PLE devices to implement 8-bit parallel CRC. The Error Flag is valid on the next clock pulse after all the data has been clocked in.

Available Software

PLEASM is a PLE Assembler written in FORTRAN 77 and available on most mini and microcomputer systems. It enables specifying PLE data in terms of logic equations (like those in Figure 19), which are assembled into a fuse-pattern format compatible with commercially available PROM programmers. PLEASM also generates a truth table from the logic equations which is later used as test vectors for logic simulation and verification. It allows complete design customization and documentation of PLE systems in a simple high-level functional language.

This CAD tool is available from Monolithic Memories at a small token charge. Please contact the software department at (408) 970-9700 extension 8130 for inquiries.

Acknowledgements

Special thanks to Frank Lee, Nadia Sachs, Vivian Kong and Vince Coli for some insightful discussions on testability, parallel CRC, and special applications of PLEs.

References

1. Birkner, Coli, Lee *"Shadow Register Architecture Simplifies Digital Diagnosis".* Monolithic Memories Application Note: AN-123.
2. Lee, Coli, Miller *"On-Chip Diagnostic Circuitry Streamlines Board and System Testing".* Electronic Design April 14, 1983.
3. Oppenhiem and Schafer *"Digital Signal Processing".* 1975 Addison Wesley Inc. Chapter 10.
4. Sachs, Nadia *"Pseudo Random Number Generator (a disguised PAL)".* Monolithic Memories Application Note: AN-118.
5. Sachs, Nadia *"Cyclic Redundancy Check Using PALs".* Monolithic Memories Application Note: AN-105.
6. Stone, Harold S. *"Discrete Mathematical Structures and Their Applications".* 1973 SRA Inc. Chapters 9.5, 9.6.
7. Vivian Kong *"Implementation of Serial/Parallel CRC Using PAL Devices".* Monolithic Memories Application Note: AN-125
8. Williams and Parker *"Design for Testability — A Survey".* IEEE Transaction on Computers, January 1982.
9. PLE Handbook, Monolithic Memories Inc. 1984.

Appendix I

Introduction to Programmable Logic — A Synopsis

A logic function, whether combinatorial or sequential, may be represented in Sum of Product (SOP) form by using De'Morgan's laws and Boolean Algebra. Any complex multi level logic function can easily be reduced to a two-level AND-OR configuration. This property of logic functions lend them a very regular character, making it possible to implement them in a structured methodical way. The uniform AND-OR array-like architecture of Programmable Logic Devices was conceived for a clean and efficient implementation of these functions, as shown in Figure 1.

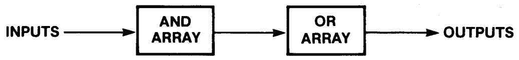


Figure 1. Structure of Programmable Logic Devices.

Either or both the arrays can be programmable, constituting three distinct families of devices as shown in Figure 2.

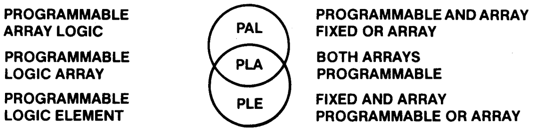

PROGRAMMABLE ARRAY LOGIC
PROGRAMMABLE LOGIC ARRAY
PROGRAMMABLE LOGIC ELEMENT
PAL
PLA
PLE
PROGRAMMABLE AND ARRAY FIXED OR ARRAY
BOTH ARRAYS PROGRAMMABLE
FIXED AND ARRAY PROGRAMMABLE OR ARRAY

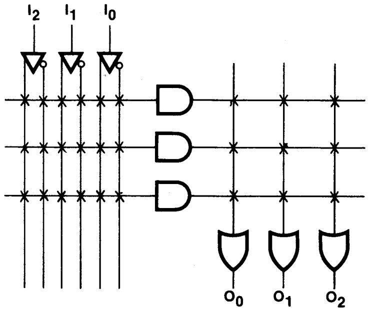
Figure 2. Structural differences between PLA, PAL and PLE.

PLA Architecture

PLAs have the most general and flexible architecture with both arrays programmable. In a (n, m, p) PLA, the AND array contains 2 x n x p programmable crosspoint connections, corresponding to n inputs and p product terms. All possible combinations of the inputs, taken together, or in groups, or even a single input, can have a product term in the AND array. The inputs not desired in a product term are disconnected, by blowing the appropriate fuse crosspoint. Usually, the number of product terms in fuse-programmable PLAs, is less than the maximum possible 2^n product terms. The OR array contains mxp programmable crosspoint connections for m outputs of the PLA. Each product term is initially connected to all the outputs, but may be disconnected, by blowing that fuse connection. This flexible architecture is most useful for implementing logic functions with a large number of product terms of varying sizes, or a large number of product terms per output.

Field Programmable PLAs are generally not very suitable to use due to their relatively poor performance. They are slower in speed compared to PALs and PLEs. A given signal must pass through two large programmable arrays, which increases the capacitance on the line, and the power dissipation of the device. For most applications, a large number of crosspoints in one or another array is usually left unused, making the architectural flexibility gained at the expense of performance, very redundant. Programming yields for PLAs are also not very impressive due to complex fuse programming circuitry required to program both arrays. Programming algorithms for field programming are also quite complicated requiring special PLA programmers.

The main advantage of PLAs is not in field programmability, but in custom VLSI circuits. Very-high-performance PLAs can be generated for those applications. Sense amplifiers on the input and output signal lines improve speed characteristics. Folding and Partitioning the arrays reduces the silicon real estate. Extensive CAD tools for functionally specifying PLAs in a high-level language, make PLAs very easy and convenient to use. Many theories for testing PLAs in VLSI have been developed and implemented, which also make PLAs attractive from the testability point of view. Special cases of these theories are also applicable to testing PALs and PLEs. PLAs are generally used to implement random logic functions and state machines in VLSI circuits.



X = PROGRAMMABLE CROSSPOINT CONNECTION

Figure 3. Functional diagram of a PLA.

PAL Architecture

PALs are the most useful and efficient of the entire family of fuse-programmable logic devices. First developed and patented by John Birkner at Monolithic Memories in 1976, the PALs have become well known for their friendliness in system configurations. The programmable AND array and fixed OR array eliminate most of the redundancies associated with PLAs. The PAL AND array is logically identical to a PLA AND array, with the only difference that PALs have fewer product terms. In the fixed OR array, each product term is connected to only one output or OR gate, which eliminates product term sharing of PLAs. The PAL configuration has permitted several architectural innovations, making the PAL family of devices extremely useful for implementing all kinds of logic functions. PAL features include outputs with/without registers that are internally fed back to the AND array, special XOR gates in the OR array, arithmetic carry generate gates in the feedback path in the AND array, programmable I/O pins, and programmable output polarity.

As the AND array in PALs is programmable, logic functions can be minimized and logic hazards removed, by combining adjacent minterms in a Karnaugh map. This group of minterms or implicants is implemented as a product term. Thus PAL outputs are glitch-free, and ideal for implementing control logic. Figure 4 illustrates the absence of hazards and race conditions in a PAL.

The disadvantage of PALs is that they have a restricted number of product terms per output, and fewer product terms in general. Certain logic functions containing a large number of product terms would require a large number of PAL devices to implement them, which increases the propagation delay, and chip count. For these applications, PLEs are ideal.

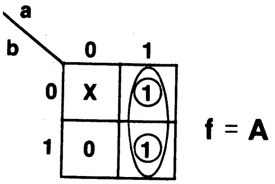


Figure 4. An arbitrary Karnaugh map for a PAL.

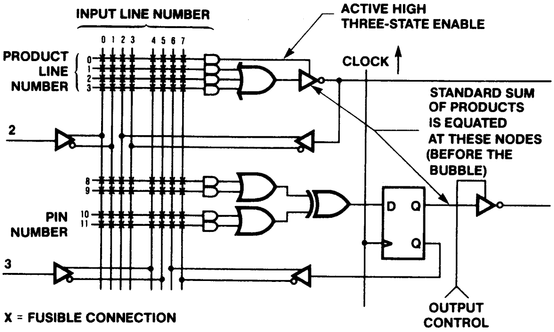


Figure 5. A section of PAL 20X8.

PLE Architecture

PLEs bridge the gap between the extreme flexibility of PLAs, and the product term restrictions in PALs. Those applications for which the PALs are not suitable, the PLEs take over. Where a PAL typically has a large number of inputs, and a small number of product terms, the PLEs have a restricted number of inputs and a large number of product terms. Similarly, a PLE has a large number of product terms per output with full product term sharing, whereas PALs have a restricted number of product terms per output with no product term sharing. Thus PALs and PLEs complement each other both structurally and functionally.

Appendix II

Some PROM/PLE applications are in the following areas:

Computer Industry

- Personal/mini/mainframe computers
- Peripheral devices
  - Disk/tape controllers
  - Video/graphics controllers
  - Smart terminal/printers
  - Stepper motor controller for disk drives
  - Cyclic Redundancy Check (CRC) codes
- Fault-tolerant self-checking computer systems

Entertainment Industry

- Video games

Communication Industry

- Signal processing in radar/speech/image
- Control of data acquizition systems
- Local area network protocol control
- Diagnostics for switching equipment
- Error detection/correction codes

Military

- Terrain data store for guidance and control of missiles
- Signal processing
- Error detection/correction systems

Medical Instrumentation

- CAT scanner

Industrial Controllers

- Valve controllers
- Measurement equipment

Appendix III

PROMs vs. EPROMs

- EPROMs are MOS circuits with very slow access times. Fastest 256 K EPROM has T_A = 200-225 nsec.
- PROMs are generally Bipolar circuits. Typical access times for 32 K PROM is between 35-50 nsec. Higher density PROMs are being developed which are expected to have T_A = 30-40 nsec.
- Low density 32 x 8 PROMs which are ideal for PLE applications, have T_A = 17 nsec. It is expected to be further reduced to 8 ~ 15 nsec.
- EPROMs store logic values as charges on MOS transistor gates which act as capacitors. This makes it susceptible to radiation. Also, EPROMs are sensitive to static and must be handled carefully.
- Bipolar PROMs use mechanical fuses to store logic values and are quite robust even in extreme operating conditions.
- EPROMs consume less power and can have much greater density.
- Quarter- and half-power bipolar PROMs are available, and 64 K and 128 K PROMs are also being developed.

IMPORTANT,
NOTES

Big, Fast and Simple — Algorithms, Architecture, and Components for High-End Superminis

Ehud Gordon and Chuck Hastings

Abstract

Density increases in today's LSI have made possible high-speed integrated circuit building blocks such as 8x8 Cray multipliers and 16x16 matrix shifters, which offer functionality beyond that attainable at any reasonable cost in yesterday's computers. These newer parts incorporate massive internal parallelism, and yet feature short logic-delay times and volume-production prices.

So it now makes good cost/performance sense to use, in high-end minicomputers consisting of only a few hundred integrated circuits, design and architectural approaches which until recently were too costly to use except in the very largest supercomputers and in specialized signal processors. In many cases, these techniques are actually the simplest and most natural ones for performing the required operations, but the block diagrams of the resulting machines look quite different from those for most recent-vintage minicomputers.

This paper presents cost/performance-effective design alternatives for conventional Von Neumann uniprocessors, based on the supercomputer design philosophy of "Big, Fast, and Simple" which is attributed to Seymour Cray. The vehicle for presenting these alternatives is a preliminary design for a multi-MIPS 64-bit floating-point TTL supermini which incorporates arrays of 8x8 multipliers and 16x16 shifters, supported by other high-speed components such as ALUs, carry bypasses, priority encoders, PALs, interface circuits, and FIFOs.

Monolithic Memories MMI

Big, Fast, and Simple — Algorithms, Architecture, and Components for High-End Superminis

Ehud Gordon and Chuck Hastings

Introduction

This paper sets forth the gist of several techniques for performing ultra-high-speed binary floating-point arithmetic: matrix shifting, normalization scanning, Cray multiplication, and division by iterated multiplication. All of these techniques achieve increased speed through extensive parallelism. Their equivalent-gate-count implications would have seemed, until quite recently, altogether too lavish to allow them to be considered for minicomputers. Now, there are some affordable higher-density and higher-speed standard semiconductor building blocks available to implement them, and they are becoming part of the minicomputer designer's everyday bag-of-tricks.

These techniques are by no means entirely new. One contemporary supermini, the Norsk Data NORD-500, is designed very much in accord with the approaches presented here[r1]. The use of shift matrices goes back at least to the Univac 1107 in the early 1960s[r2], and the use of priority encoders to do high-speed parallel floating-point normalization scanning goes back to 1972 if not earlier[r2]. Cray (combinatorial) multiplication was used in a specialized Control Data system delivered about 1960[r3], and one type of division algorithm based on iterated multiplication was used in the IBM 360/91 about 1966 or 1967[r4].

Here, all of these techniques are collected together in one brief tutorial paper, and are described from the viewpoint of the logic designer rather than from that of the system architect. A specific implementation for each technique is detailed, using the latest, densest, and swiftest standard TTL-compatible semiconductor components. In concert, these techniques provide an itegrated design approach suitable for the next generation of minicomputers. The treatment is necessarily abbreviated, but this paper is in fact a condensed version of a much longer applications report[r5]

Design Context

Rationale for Design Choices

The context for discussing algorithms and design approaches here is a high-end, TTL, floating-point Von Neumann uniprocessor having a simple floating-point data format chosen to be very similar to many familiar formats, but not exactly like any of them. Some of the similarities and differences are discussed below. This choice is for tutorial reasons; it allows us to explain arithmetic techniques without getting bogged down in divisive and controversial side issues, in such a way that any experienced computer architect/designer can readily see what small changes and corrections would have to be made in order for these techniques to work under different design conditions and data formats. Although our data format would doubtless be satisfactory enough if it really were used in an actual computer, we did not choose it for that reason, but rather to be a kind of design-philosophical weighted average of many of the formats which are significant in the marketplace.

System Architecture

Likewise, whatever assumptions we have made about systems architecture are for reasons which are more tutorial than technical. We present one basic design alternative in each section, which tends to be a speed-optimizing rather than a cost-optimizing alternative; other possibilities are mentioned in passing. In some cases, logic has been duplicated outright in order to gain speed while keeping the architecture as simple as possible. The reader can trade off performance and package count to adapt the design to differing system requirements.

Our floating-point processor design is partitioned into two independent modules: a floating-point adder/subtracter unit, and a floating-point multilier/divider unit. In a lower-cost design, certain hardware entities which we have shown as separate could be merged; in a highest-end design, multiple modules of each of these types might be used together with some sophisticated "dispatching" hardware, an approach reminiscent of the Control Data 6600. Past experience has shown that implementing the four basic operations of addition, subtraction, multiplication, and division is responsible for most of the cost of floating-point processors, and that the technical approach taken to these four operations dictates the overall architecture. We have, however, also addressed high-speed implementations of fixed-point-to-floating-point conversion, floating-point-to-fixed-point conversion, and square root.

All of the different types of TTL LSI, MSI, and SSI components required to implement this design either are commercially available right now, or else are currently being designed at Monolithic Memories with introduction planned to occur within a few months.

Floating-Point Data Format

The design techniques we present are useful with virtually any of the mainstream floating-point data formats, and it is not our intention to promote any particular format or number-representation scheme at the expense of any other. As far as we know, our format has not been used in any actual computers. However, it uses exactly the same partitioning of a 64-bit data word into fields as is used in the 64-bit form of the IEEE standard for microprocessor floating-point arithmetic (see Figure 1), although — unlike the IEEE format — our format does not use any "hidden bit" and hence potentially allows for the inclusion of unnormalized floating-point arithmetic operations. In any case, for both our format and the IEEE format, the leftmost or most-significant bit in bit-position 63 is the sign of the mantissa, an 11-bit characteristic occupies bit-positions 62-52, and a 52-bit mantissa occupies bit-positions 51-00. The characteristic is interpreted as a "biased" number representing a power of 2.

SIGN	CHARACTERISTIC	MANTISSA
63	62 52	51 00

Figure 1. Floating-Point Data Format.

A few definitions to clarify various floating-point concepts are appropriate. The meaning of "characteristic" is "binary exponent," but with a special representation — the coding of the sign bit is reversed, or equivalently the entire field may be considered as a twos-complement 11-bit number with a "bias" of 10000000000 (the range midpoint) added to it. The characteristic 00000000000 represents a binary exponent of -1024 — that is, the entire floating-point number represents 2^{-1024} times the mantissa. A number actually between ½ and 1 has a representation with a characteristic of 10000000000, which implies that the mantissa is taken times 2^0, and so forth. All mantissas are assumed to be *normalized* — there is always a 1 in bit position 51, and thus the mantissa considered by itself without being multiplied by the implied binary exponent is between ½ and 1.

It may at times be felt desirable by computer architects and users to be able to use *unnormalized* numbers which do not follow this restriction, for reasons too complex to be considered here. Other architects and users deny this need. If there is never any need to represent unnormalized numbers, then bit position 51 has effectively gone to waste — it always contains 1, and hence conveys no new information not already implied by the interpretation of the entire 64 bits as a floating-point number. Some computer architectures thereby employ a "hidden bit," which is catenated to the mantissa at the left and whenever the entire floating-point number is "unpacked" or separated into its constituent parts for computational operations. This "hidden bit" is always understood by the hardware to be 1. The tradeoff is that one more bit of mantissa accuracy is gained, but that unnormalized floating-point numbers become unrepresentable.

If the sign of the mantissa is at the very left, the characteristic is "biased" and in the middle, and the mantissa is on the right, it can be shown that — even though the result of the operation is garbage — *fixed*-point subtraction or comparison of two floating-point numbers always gives the right answer as to which of them is larger. To put the matter another way, under these assumptions the "partial ordering" of the set of floating-point numbers does not differ from that of the set of fixed-point numbers represented using the same number of bits. Because of the obvious implied hardware economies, "biased" representation of floating-point characteristics has been widely used for three decades.

In many computer architectures, the characteristic is interpreted to imply a power of 8 or 16 rather than a power of 2; 8 has been used by Burroughs and Ferranti, and 16 by a large number of firms. The "long" floating-point format used in the IBM 360/370 computers has a 7-bit power-of-16 characteristic and a 56-bit mantissa. It may be seen that

$$16^x = (2^4)^x = 2^{4x}$$

so that the range of values of the characteristic is almost (albeit not quite) as great as if it were represented using two more bits and were interpreted as a power of 2 rather than of 16. On the other hand, "normalization" now implies that the *hexadecimal* digit occupying bit positions 55-52 must be nonzero, but it can be as small as 0001 if examined as a binary 4-bit number. Hence the mantissa, looked upon as a 56-bit binary number, can have up to 3 leading zeroes, and is really only equivalent to a 53-bit mantissa in a representation using a power-of-2 characteristic. Thus this scheme is closely similar to the use of a 9-bit power-of-2 characteristic and a 53-bit mantissa, and there has been a net loss of one bit's worth of information-carrying capacity. There is still, of course, the sign of the mantissa in bit position 63 as always. The great advantage of the scheme is that the sign of the mantissa plus the 7-bit characteristic occupies exactly one byte, which offers major simplifications in the data paths within a processor. For this reason, the scheme has been used by IBM and all of the other firms who have produced reverse-engineered IBM 360/370-lookalike computers — Amdahl, IPL, Magnuson, Nanodata, National Advanced Systems, Two-Pi, and others — as well as firms using other architectures such as Data General, Interdata, SEL, and Xerox.

Floating-point data formats can be sign-magnitude, twos-complement, or ones-complement, just as fixed-point data formats can. Today, sign-magnitude seems to be the commonest, but both of the others are still alive. In either of the complementary representations, when the number is negative and the mantissa therefore gets complemented, the characteristic must also get complemented in order to preserve the previously-mentioned "partial-ordering" property and allow the use of fixed-point hardware to perform floating-point comparison.

FORMAT	FIELDS	BASE	-#	H-BIT	F/I
One used in this paper	1-11-52	2	SM	No	F
IEEE 64-bit	1-11-52	2	SM	Yes	F
NORD-500 (Long)	1-9-54	2	SM	Yes	F
DEC PDP-11 (Long)	1-8-55	2	SM	Yes	F
HP-3000 (Long)	1-9-38	2	SM	Yes	F
Cray-1	1-15-48	2	SM	No	F
CDC 6600/7600 (and "Cyber" series)	1-11-48	2	1C	No	I
IBM 7030 ("Stretch")	1-11-48-4	2	SM	No	F
FPS AP-120B	1-10-27**	2	2C	No	F
IBM 360/370 (Long) (and Amdahl, etc.)	1-7-56	16	SM	No	F
SEL, Xerox (Long)	1-7-56	16	2C	No	F

NOTES:

"1-11-52" under FIELDS implies one sign bit, 11 characteristic bits, and 52 mantissa bits. In the case of the IBM 7030, there are also 4 flag bits. **In the case of the Floating Point Systems AP-120B array processor, the sign bit is positioned immediately before the mantissa.

"-#" means the scheme for representing negative numbers — *S*ign-*M*agnitude, *1*s-*C*omplement, or *2*s-*C*omplement.

"H-BIT" means whether or not a hidden bit is used.

"F/I" means whether the mantissa is interpreted as a *F*raction or as an *I*nteger.

Certain of the above machine architecture also include "short-format" floating-point as an alternative, with 32 bits used to represent a complete (albeit not very precise) floating-point number. Obviously, an arithmetic unit which can handle the corresponding long format can also handle the short format, if provision is made for the abbreviated mantissas which thereby ensue.

Table 1. Various Floating-Point Formats

Most floating-point formats are "fractional," which is to say that mantissa are interpreted as falling in the range ½ to 1 (1/16 to 1 for power-of-16-characteristic formats). A few are "integer," which implies that mantissas are interpreted as falling in the range ½(2^n) to 2^n-1 for n-bit mantissas. Other variations include the provision for representation of very small unnormalized ("denormalied") numbers in otherwise hidden-bit formats such as those in the IEEE standard; a different ordering of the mantissa sign, mantissa value, and characteristic; and the use of exponents represented as sign-magnitude or twos-complement rather than biased numbers.

With this framework for understanding floating-point formats, several significant ones are listed below along with the one used in this paper, to provide insight into the effect of a change of format on the properties of the hardware and algorithms to be described.

There are some "concealed" ways in which floating-point formats can vary further. For instance, rounding algorithms can vary; the "unbiased" algorithm is used here. Also, the number of extra or "guard" bits carried along within the processor for intermediate results during floating-point operations can vary from 4 to 48; 12 guard bits are used here.

Addition and Subtraction

Algorithm

The algorithms for sign-magnitude addition and sign-magnitude subtraction are virtually identical. Subtraction requires that the sign bit of the operand being subtracted be complemented, and this step may be entirely implicit and is simply omitted for addition. For either addition or subtraction, the situation must be reduced to two cases; addition of operands with like signs, and addition of operands with unlike signs.

The characteristics of both operands must be made equal in order to line up both mantissas so that they can be correctly added. This alignment is accomplished by shifting the mantissa of the smaller operand right, while increasing its characteristic until both characteristics are equal. In the floating-point adder/subtractor unit shown in Figure 2 below, the "shift-amount generator" determines which characteristic is smaller and thereby selects the mantissa to be realigned; if the two characteristics are equal, the output of the "magnitude comparator" is used to break the tie. The "SE" path (*S*hift *E*nable) is used to force one of the two right shifters to perform a shift operation of the number of places determined by the "shift-amount generator," and the other right shifter executes a "bypass" or zero-place-shift operation.

The two mantissas are now positioned to enter the "carry-lookahead adder" properly with respect to each other. This adder performs an add operation for the add-with-like-signs case, or a subtract operation for the add-with-unlike-signs case. Since the smaller operand is always the one which gets complemented in the unlike-signs case, recomplementation of the result mantissa is never necessary.

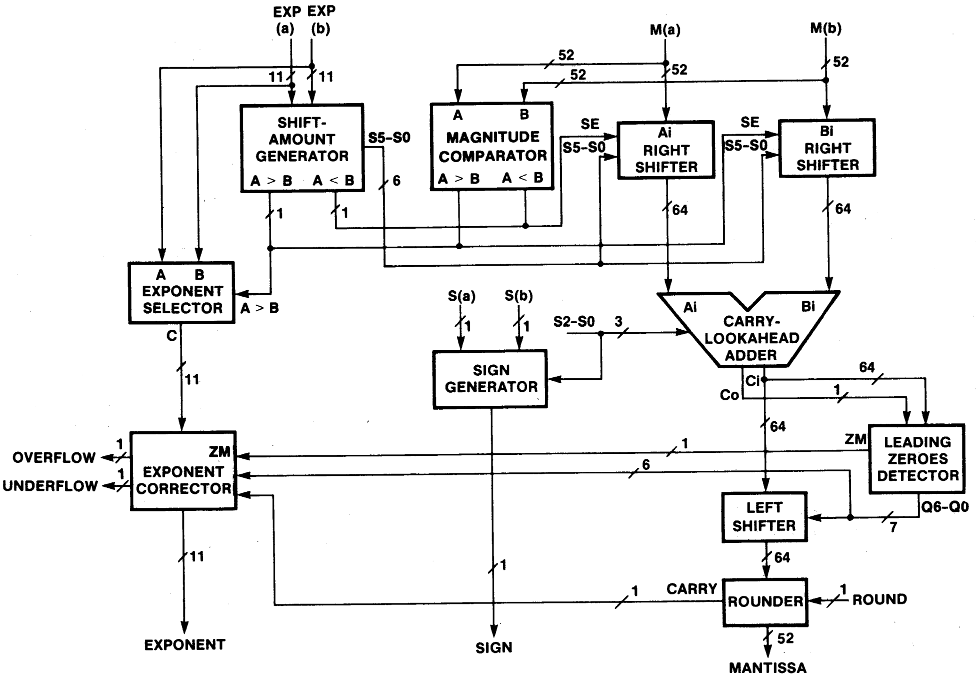


Figure 2. Floating-Point Adder/Subtracter Unit

The result mantissa is scanned by the "leading zeroes detector" to see how many bit positions it must be left-shifted in order to renormalize it. This detector controls the left shifter, and so the result mantissa emerges from the latter positioned properly to be normalized in the final result. At this stage, the result mantissa is still a 64-bit number; the 12 least-significant bits are the "guard bits." The final step in processing the result mantissa is then to inspect the guard bits to see if the result mantissa must be incremented by 1 in its least-significant final bit position ("rounding up"), or if it can remain as it is.

The sign bit of the result is formed by the "sign generator," which is essentially one specialized gate. The characteristic of the result is the characteristic of whichever of the original operands was larger, adjusted up by 1 or else down if the result mantissa came out with a different number of significant bits than the mantissa of the larger of the original operands had — obviously 52 bits in this context. If the result exponent thereby winds up outside the range of numbers which can be represented in an 11-bit biased field, floating-point overflow or floating-point underflow has occurred and the corresponding indication is made.

In Figure 2, the two original operands are denoted as a and b. During the algorithm, the characteristics are temporarily converted to exponents, denoted as EXP(a) and EXP(b), so that they can be operated upon using twos-complement arithmetic according to the usual rules. S(a) and S(b) are the respective sign bits, and M(a) and M(b) are the respective mantissas.

Hardware

The shift-amount processor includes two 12-bit parallel adders, each consisting of three 74S382A ALUs connected in ripple-carry mode; for word lengths of at most 12, this approach is as fast as using carry-bypass circuits. One of these 12-bit adders computes EXP(b) - EXP(a) while the other computes EXP(a) - EXP(b), and so one the adders always yields a positive number which can be used as a shift count. The selection of the appropriate set of adder outputs is made using quad 2-input muxes such as 74S257s, controlled by the sign of an adder result.

The magnitude comparator is a 52-bit array of 74S85 comparators, cascaded as shown in Texas Instruments' data sheet for this part[r6]. Alternatively, a 64-bit array may be used, and placed earlier in the data stream, since because of the use of biased characteristics in this floating-point format a broadside fixed-point comparison may be used to determine which of the two floating-point operands a and b is larger.

The right shifters are arrays of 74S530 16x16 LSI shifters, organized as shown in Figure 3. The 74S530 is a 48-pin part, capable of shifting in either direction, and featuring three-state outputs which are controlled jointly by the output-enable inputs and the 4-bit shift count when a shifting operation is being performed. The incomplete 4x4 array of Figure 3 is configured to perform right shifts only; a complete 4x4 array such as that of Figure 4 can perform either right or left shifts; and Figure 5 shows another incomplete 4x4 array, this time configured to perform left shifts only. In each case, however, the array can perform any specified shift of from 0 to 63 bit positions on a one-level flow-through basis in the same amount of time (20 nanoseconds worst-case). The logic of the 74S530 operates to ensure that exactly one output, of the appropriate shifter in the array, is enabled to drive each bit position of the output bus. Since the number system is sign-magnitude here, the 74S530's capability of filling vacated operand positions with copies of the sign bit is not used.

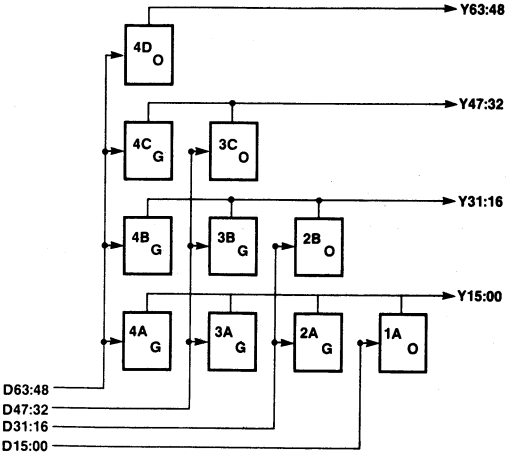


Figure 3. One-Level Right Shifter Using 74S530s

The position of each 74S530 in the array is denoted in Figures 3, 4, and 5 by a code consisting of a number and a letter: the number signifies which group of 16 bits on the input bus is connected to the inputs of that 74S530, and the letter signifies to which group of 16 bits on the output bus the outputs of that 74S530 go. For instance, 74S530 #3B has its inputs connected to bit positions 47-32 on the input (D) bus, and its outputs connected to bit positions 31-16 on the output (Y) bus.

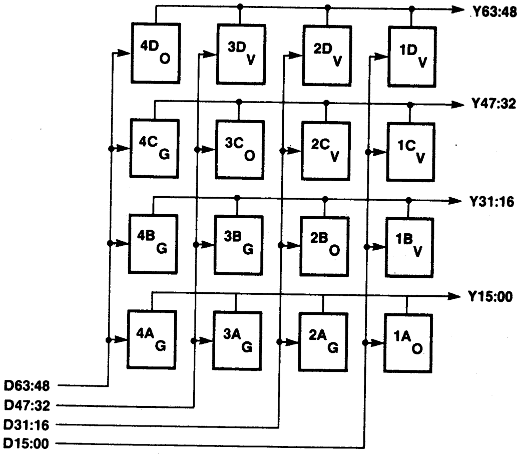


Figure 4. One-Level Bidirectional Shifter Using 74S530s

The 74S530 has a special control pin to tell it what its position is within an array such as one of these. This "location identifier" (LID) pin may be grounded (G) to indicate to a 74S530 that it is to the left of the "home" diagonal consisting of 74S530s #4D, #3C, #2B, and #1A; or tied to V_{CC} (V) to indicate that it is to the right of the home diagonal. The 74S350s right on the home diagonal have this input left open (0), which causes it to register the intermediate state of +2.5V because of a voltage divider circuit within the chip between V_{CC} (+5V) and ground. The significance of the "home diagonal" is of course that 74S530s on it are the ones used when a direct pass-through or "bypass" (zero-place shift) operation is performed.

These shifter configurations can also be implemented as three-level arrays of 74S350 (Am25S10) 4x4 MSI shifters, at a considerable increase in parts count. For the unidirectional configurations of Figures 3 and 5, 48 74S350s are required in each case as compared with 10 74S530s. For the bidirectional configuration of Figure 4, 96 74S350s are required as compared with 16 74S530s. This disparity would be aggravated if a complementary number system were used, because additional external logic would be needed to perform the sign-bit-fill ("sign-extension") operations which the 74S530 is capable of doing automatically.

The operation-code table for the 74S530 will give some idea of its capabilities:

OPCODE	OPERATION PERFORMED IF LID PIN IS "OPEN"		ALTERNATE OPERATION	
I I I 2 1 0	MN.	NAME	LID V_{CC}	LID GND
0 0 0	LSH	Left SHift	LRT	FIL
0 0 1	LRT	Left RoTate	LRT	LRT
0 1 0	BIT	Bit InserT**	ZZZ	ZZZ
0 1 0	BYP	BYPass**	ZZZ	ZZZ
0 1 1	FIL	FIL1	ZZZ	ZZZ
1 0 0	RSC	Right Shift with twos-Complement shift count	FIL	RRC
1 0 1	RRC	Right Rotate with twos-Complement shift count	RRC	RRC
1 1 0	RSH	Right SHift	FIL	RRT
1 1 1	RRT	Right RoTate	RRT	RRT

NOTES:

**When the opcode I_{2-0} is 0 1 0, another input control line (PI, for Polarity/Insertion) determines whether a BIT operation is performed. For all other possible values of the opcode field, PI determines whether or not the data outputs should be complemented.[r7] Notation here is assertive-high — 0 means Low.

Left shifts assume that the shift-count field is a positive number. There are two alternative sets of right shifts, which differ in that one set interprets the shift-count field also as a positive or "sign-magnitude" number (opcodes 1 1 0 and 1 1 1), whereas the other set interprets this field as a twos-complement number (opcodes 1 0 0 and 1 0 1). The designer can use either set of right-shift operations at his convenience. The reasons for preferring one set or the other in specific cases are outside the scope of this paper.

The LID control input causes shifters not on the home diagonal to execute different operations when certain opcodes are applied. The total effect is that the entire array of shifters executes a 64-bit operation equivalent to the 16-bit operation which the parts on the home diagonal (LID open) execute for that opcode value.[r7] "ZZZ" means that all data outputs go to hi-Z state. The LRT and RRC operations actually are identical, but arise from different circumstances.

Table 2. 74S530 Operation Code

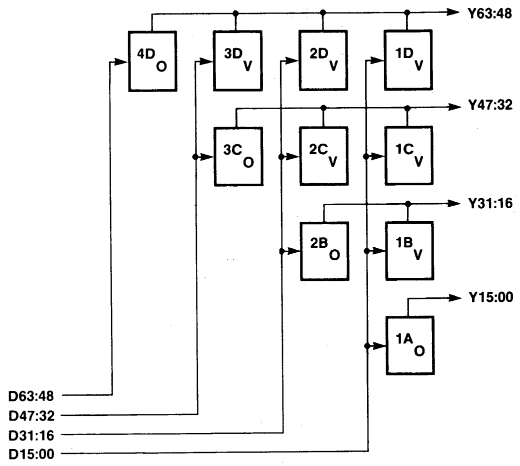


Figure 5. One-Level Left Shifter Using 74S530s

The carry-lookahead adder is comprised of 16 4-bit arithmetic logic units (ALUs) and 5 4-group carry bypasses. 15 of the ALUs are 74S381As, but the most-significant one is a 74S382A in order to provide explicit left-end carry-out and overflow indications. The carry bypasses are 74S182s. In this design, 74S181s could not be used because they lack the "inverse subtract" (B-A) operation, which is needed as well as the usual "subtract" (A-B) operation in order to be able to always complement the *smaller* of the two input operands in the add-with-unlike-signs case, as previously discussed. The 74S381A and 74S382A are slightly faster than the 74S181, unlike the standard 74S381 which is slower; all of these parts can perform both subtraction and inverse subtraction. The only difference between the 74S382A and the 74S381/1A in function is that the 74S382A has carry-out and overflow outputs in place of the propagate and generate outputs of the 74S381/1A, which are used to drive the inputs of 74S182s.

As has already been mentioned, for adders of length up to 12 bits essentially equal performance can be obtained by connecting 74S382As one to another directly in a ripple-carry configuration. For longer adders, performance is very much better if 74S381As are used instead, in groups of four. Within each group, a 74S182 is used to supply the input carries for all 74S381As except for the least-significant one. A fifth 74S182 is used to tie these four groups together.[r8]

Adders can be constructed in this manner for any word length which is a multiple of 4. The add/subtract time for such an adder increases only very slowly as the word length increases — essentially, quadrupling the word length introduces two additional logic delays. The reason for choosing the word length to be 64 in this design context is the presumption that 64-bit fixed-point addition and subtraction capabilities are an architectural requirement for the computer, and can most economically be performed in this unit rather than somewhere else.

Figure 6b shows the least-significant group of eight ALUs, all 74S381As in this case, which covers bit positions 31-00. The most-significant group shown in Figure 6a has a 74S382A as its most-significant ALU as already mentioned. The 74S182 shown in Figure 6a does not receive propagate and generate

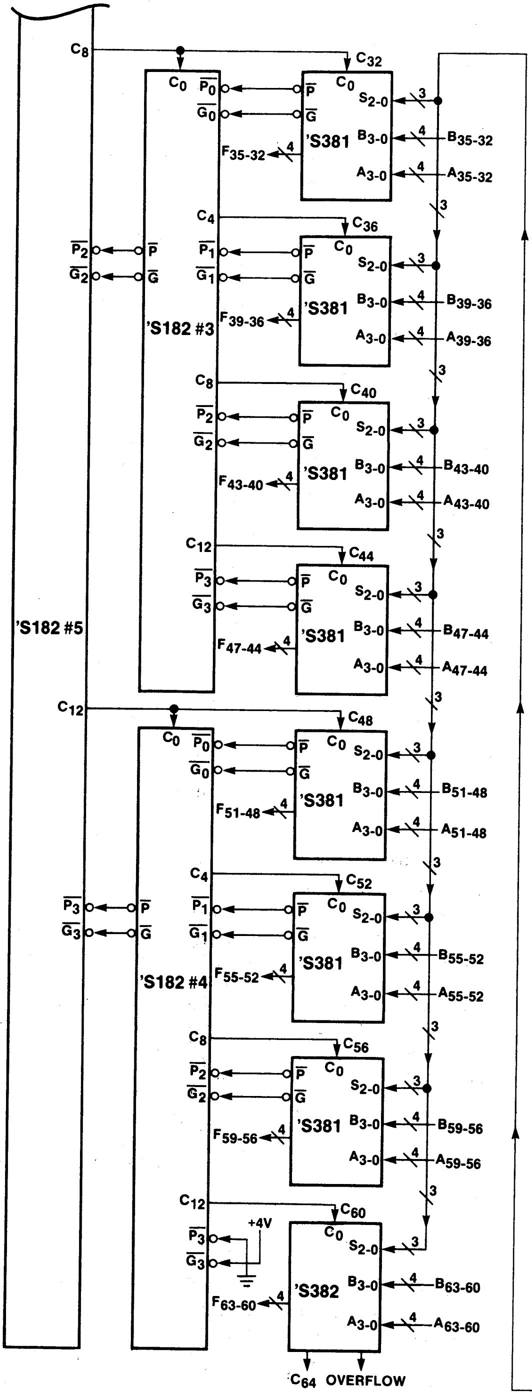


Figure 6a. Most-Significant ALU Group

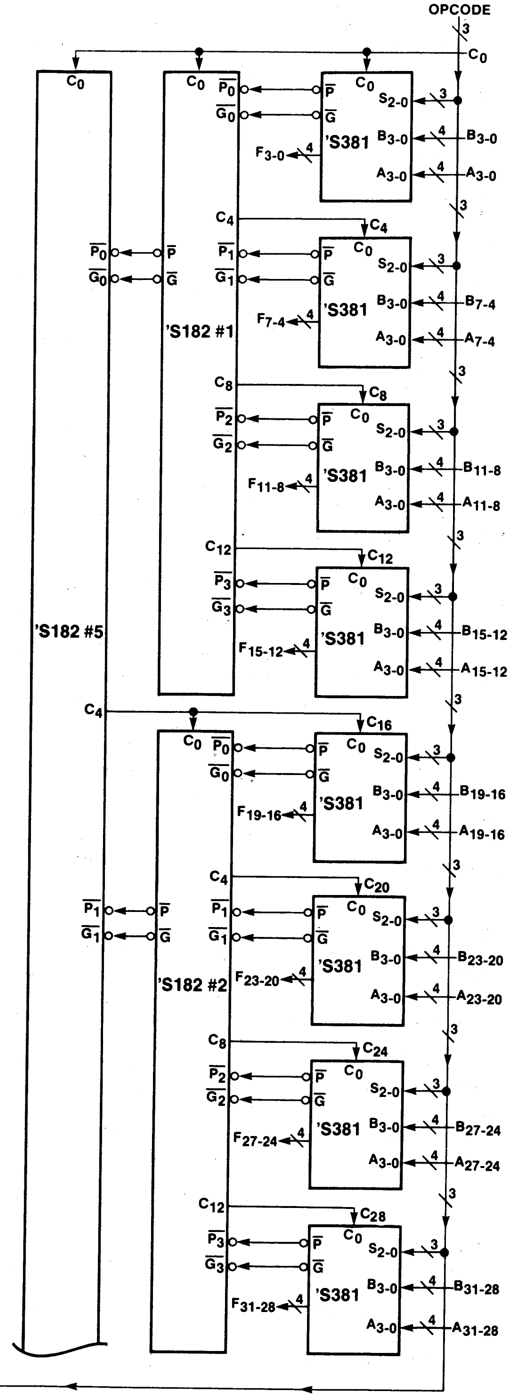


Figure 6b. Least-Significant ALU Group

inputs from this most-significant ALU, but that does not matter since these inputs would affect only its propagate and generate outputs to a higher level, and since there are only two levels of 74S182s in this adder no 74S182 *carry* outputs can be affected by this omission.

The following is the operation-code table for these ALUs, in assertive-high notation:

OPCODE S S S 2 1 0	ARITHMETIC OR BOOLEAN OPERATION
0 0 0	Clear (force all data outputs low)
0 0 1	Inverse Subtract (B-A)
0 1 0	Subtract (A-B)
0 1 1	Add (A-B)
1 0 0	Exclusive-OR ((A&$\overline{B}$)v($\overline{A}$&B))
1 0 1	OR, i.e., Inclusive-OR (AvB)
1 1 0	AND (A&B)
1 1 1	Preset (force all data outputs high)

Table 3. 74S381/1A and 74S382/2A Operation Codes

The leading-zeroes detector is a maximum-speed-configuration 64-bit priority encoder, made up of 74S148 or 74S348 8:3 priority encoders and 74S151 or 74S251 8:1 multiplexers ("muxes"). The choice of which part of each type to use here is not a major issue; the 74S148 and the 74S151 have standard totem-pole outputs, whereas the 74S348 and 74S251 have three-state outputs, and there are no other functional differences in either case. In this configuration, the three-state behavior is not required, and the parts are always enabled. However, three-state parts may still be preferable because of their improved drive capabilities if there is any appreciable physical distance between driven inputs and driving outputs, or many other loads are driven besides the ones required strictly for this application.

Priority encoders are not usually considered to be arithmetic components. Their original interrupt-request lines should be given priority and be recognized first. The 74S148 and 74S348 are also fast enough to be useful in determining which of several competing requests for control of a digital-system bus should be granted first, and for other such resource-allocation applications. However, the examination of an unnormalized floating-point mantissa which is the result of an add or subtract operation, to determine how many leading zeroes it has and therefore how many places it must be left-shifted in order to normalize it, reduces to exactly the same logic.

In order to optimize the internal speed of the part, TTL priority encoders have been designed as intrinsically assertive-low devices, beginning with the original Fairchild 9318 more than a decade ago. That is, they are designed to look for the first Low in a field of Highs, rather than the other way around. Consequently, in this design the mantissa must be presented to the array of encoders as an assertive-low word — 1 is represented by Low, and 0 by High, in contrast with the assertive-high notation used in Tables 2 and 3. No extra components are actually required to render the ALU outputs as a 64-bit assertive-low word, if the system is designed according to the block diagram of Figure 2. Since the 74S530 shifters have controllable output polarity they can present their outputs assertive-low to the ALU inputs, even if they themselves see assertive-high data at their inputs. And the 74S381/1A and 74S382/2A, like the 74S181, have internal dual symmetry such that they can operate correctly with either assertive-high or assertive-low data, and the assertiveness of the result is the same as that of the input operands. Other system designs may place the leading-zeroes detector right at the outputs of a shifter array, and the same situation would continue to hold.

Table 4 summarizes the functioning of the 74S148 and 74S348. To avoid any further confusion as to assertiveness, "H" and "L" are used this time for "High" and "Low" respectively, rather than "1" and "0" as in Tables 2 and 3. In Table 4, "X" means "don't care" — this input exerts no control over any of the outputs, because a higher-priority input is being asserted. Also, "H/Z" for an output means that in the totem-pole 74S148 it is High, whereas in the three-state 74S348 it is in the Hi-Z (high-impedance) state.

INPUTS									OUTPUTS				
$\overline{EI}$	$\overline{I}$ 0	$\overline{I}$ 1	$\overline{I}$ 2	$\overline{I}$ 3	$\overline{I}$ 4	$\overline{I}$ 5	$\overline{I}$ 6	$\overline{I}$ 7	$\overline{A}$ 2	$\overline{A}$ 1	$\overline{A}$ 0	$\overline{GS}$	$\overline{EO}$
H	X	X	X	X	X	X	X	X	H/Z	H/Z	H/Z	H	H
L	H	H	H	H	H	H	H	H	H/Z	H/Z	H/Z	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

Table 4. Function Table for 74S148 and 74S348

To readily understand Table 4, and relate it to Figure 7 which shows the entire 64-bit leading-zeroes detector, it is best to start with the idea that *all* inputs and outputs of these encoders are assertive low — even the code outputs $\overline{A}_{2-0}$ — and simply note that the manner in which the data inputs $\overline{I}_{7-0}$ have been numbered implies that the highest-priority input forces an assertive-low 7 (L L L) and the lowest-priority input forces an assertive-low 0 (H H H). For the totem-pole version, if no inputs are asserted (second line of Table 4) or if the Enable Input goes High and disables the part, the outputs go to the 0 condition (H H H); for the three-state version, the outputs go to Hi-Z state in exactly these cases. The Group Select output is an assertive-low OR of the eight priority inputs, and the Enable Output is its logic inverse, except that both of these go High when the part is disabled.

The cascaded arrangement of Figure 7 uses 8 encoders at the first level, and one encoder at the second level which examines the Group Select outputs from the first level. The address of the most-significant of the 64 data bits which is Low is available as a 6-bit number; the least-significant 3 bits are from one of the 8 first-level encoders, and the most-significant 3 bits are from the second-level encoder and are also the address of the proper first-level encoder from which the least-significant 3 bits should be read. The second-level Group Select output is in effect a 64-

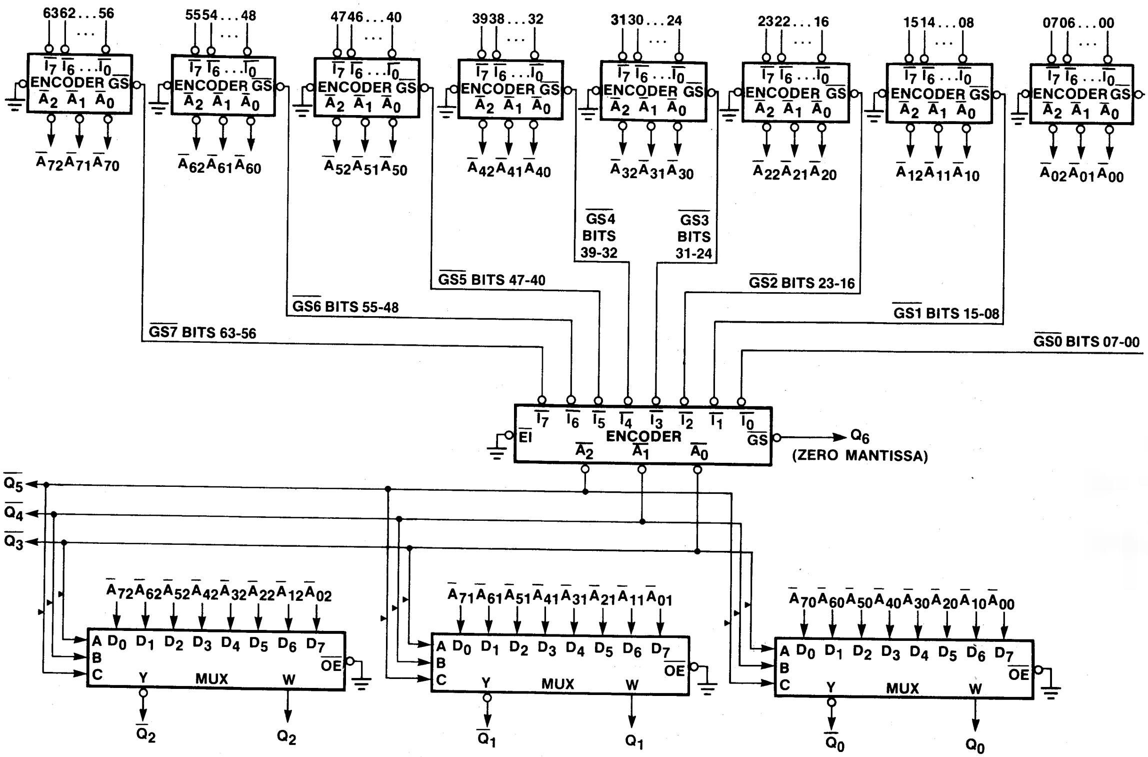

6362...56
5554...48
4746...40
3938...32
3130...24
2322...16
1514...08
0706...00
I7 I6 ... I0
ENCODER GS
A2 A1 A0
$\overline{A}_{72}\overline{A}_{71}\overline{A}_{70}$
$\overline{A}_{62}\overline{A}_{61}\overline{A}_{60}$
$\overline{A}_{52}\overline{A}_{51}\overline{A}_{50}$
$\overline{A}_{42}\overline{A}_{41}\overline{A}_{40}$
$\overline{A}_{32}\overline{A}_{31}\overline{A}_{30}$
$\overline{A}_{22}\overline{A}_{21}\overline{A}_{20}$
$\overline{A}_{12}\overline{A}_{11}\overline{A}_{10}$
$\overline{A}_{02}\overline{A}_{01}\overline{A}_{00}$
$\overline{GS4}$ BITS 39-32
$\overline{GS3}$ BITS 31-24
$\overline{GS5}$ BITS 47-40
$\overline{GS2}$ BITS 23-16
$\overline{GS6}$ BITS 55-48
$\overline{GS1}$ BITS 15-08
$\overline{GS7}$ BITS 63-56
$\overline{GS0}$ BITS 07-00
I7 I6 I5 I4 I3 I2 I1 I0
EI ENCODER GS
A2 A1 A0
Q6 (ZERO MANTISSA)
$\overline{Q_5}$
$\overline{Q_4}$
$\overline{Q_3}$
$\overline{A}_{72}\overline{A}_{62}\overline{A}_{52}\overline{A}_{42}\overline{A}_{32}\overline{A}_{22}\overline{A}_{12}\overline{A}_{02}$
$\overline{A}_{71}\overline{A}_{61}\overline{A}_{51}\overline{A}_{41}\overline{A}_{31}\overline{A}_{21}\overline{A}_{11}\overline{A}_{01}$
$\overline{A}_{70}\overline{A}_{60}\overline{A}_{50}\overline{A}_{40}\overline{A}_{30}\overline{A}_{20}\overline{A}_{10}\overline{A}_{00}$
A D0 D1 D2 D3 D4 D5 D6 D7
B OE
C Y MUX W
$\overline{Q_2}$
Q2
$\overline{Q_1}$
Q1
$\overline{Q_0}$
Q0

Figure 7. Detailed Logic Design of Leading-Zeroes Detector

bit assertive-low OR of all of the 64 data bits, since all 9 encoders in the arrangement have their Enable Inputs grounded so that they are permanently enabled, as do the muxes.

Although 8:1 muxes are the fastest approach to obtaining the least-significant 3 bits of the 6-bit address, there are some others which readers of priority-encoder data sheets[r6, r9] and other applications literature[r10] may encounter. These are generally substantially slower and do not offer sufficient savings in parts count to make up for the loss in speed. One approach is to use a decoder such as a 74S138 so that the 3-bit code from the second-level encoder can select one of the 8 Enable Inputs in the rank of first-level encoders. If the latter are three-state parts, their code outputs can be bussed together; if they are totem-pole parts, some form of 8-input AND or NAND gate is needed to function as an assertive-low OR to tie the code outputs together for each bit position of the 3-bit code. The slowest and most economical approach is to daisy-chain encoders together, with the Enable Output of one driving the Enable Input of the next, and the outputs likewise bussed or ORed together depending on whether three-state or totem-pole parts are being used.

For the IBM 360/370 floating-point data format, or for any other format based on powers of 4, 8, 16, etc., the design of the leading-zeroes detector changes appreciably. For the IBM 360/370 format, the scan is no longer for the most-significant nonzero *bit* — rather it is for the most-significant nonzero *4-bit group.* (The term "nybl" is often used for a 4-bit group, in analogy to "byte.") An efficient design would use a 4-input NOR gate looking at the 4 data bits of each nybl of the mantissa, and just two 74S348s comprising a 16-bit priority encoder to scan the outputs of all 16 NOR gates which would be needed to scan a 56-bit result mantissa with the usual 8 guard bits. Here, the simple daisy-chain arrangement just described is the fastest approach, since there are only two encoders and only one level of encoders. The Group Select output of the most-significant encoder can be used as the most-significant bit of the 4-bit code. In this design context, by the way, 4 74S530s suffice to obtain complete bidirectional shift capability, but with shift distances restricted to be multiples of 4 places. Inputs and outputs of these 74S530s are interleaved in such a way that each nybl of the input word, and also the output word, has one bit connected to each of the 4 74S530s in sequence.

The rounder consists of another full-length adder like the carry-lookahead adder just described, plus a PAL™ to control when rounding up is to occur. If this adder has absolutely no other function within the system than rounding up, it could in principle be only a 52-bit adder, but we have shown it as a 64-bit adder which would be equally fast. Actually, to round up, a "half adder" is all that is necessary, but the semiconductor industry has up until now not provided dedicated high-speed half-adder devices. The rounding-control PAL operates according to the following function table:

DATA INPUTS													
D 12	D 11	D 10	D 09	D 08	D 07	D 06	D 05	D 04	D 03	D 02	D 01	D 00	ROUNDING OPERATION
X	0	X	X	X	X	X	X	X	X	X	X	X	Truncate
0	1	0	0	0	0	0	0	0	0	0	0	0	Truncate
1	1	0	0	0	0	0	0	0	0	0	0	0	Round Up
X	1	1	X	X	X	X	X	X	X	X	X	X	Round Up
X	1	X	1	X	X	X	X	X	X	X	X	X	Round Up
X	1	X	X	1	X	X	X	X	X	X	X	X	Round Up
X	1	X	X	X	1	X	X	X	X	X	X	X	Round Up
X	1	X	X	X	X	1	X	X	X	X	X	X	Round Up
X	1	X	X	X	X	X	1	X	X	X	X	X	Round Up
X	1	X	X	X	X	X	X	1	X	X	X	X	Round Up
X	1	X	X	X	X	X	X	X	1	X	X	X	Round Up
X	1	X	X	X	X	X	X	X	X	1	X	X	Round Up
X	1	X	X	X	X	X	X	X	X	X	1	X	Round Up
X	1	X	X	X	X	X	X	X	X	X	X	1	Round Up

Table 5. Function Table for Rounding-Control Logic (Implemented Using PAL16C1)

The rounding algorithm implemented by the logic of Table 5 is "unbiased" rounding, so called because on a statistical basis it makes the error due to rounding turn out to be positive in 50% of all possible cases and negative in 50% of all possible cases, so that in some sense the total error in all possible cases is zero. A verbal description of the algorithm is that the final result operand (here in bit positions 63-12) is always truncated, i.e., not rounded up, if bit position 11 contains a 0. Bit positions 11-00 here are, of course, the guard bits. If bit position 11 contains a 1, rounding up occurs if *any* other guard-bit position also contains a 1. If that rule does not suffice because bit position 11 is the *only* guard-bit position containing a 1, then the rule is followed to always make the final 52-bit result an even number — that is, round it up if and only if it is odd, i.e., if bit position 12 contains a 1. Unbiased rounding is a relatively elegant approach, generally quite in favor with numerical analysts concerned with using computers to solve large problems, but often avoided by logic designers in the days before PALs because it was considered too costly in hardware as compared to cruder algorithms.

Incidentally, the reason here for choosing 12 guard bits is simply the presumption that the main carry-lookahead adder should be 64-bit in order to cope with required fixed-point addition and subtraction operations, which means that 12 bits are left over in obtaining a 52-bit mantissa in floating-point addition and subtraction. 8 guard bits are used today in IBM 360/370 computers, probably with a similar rationale, although some early models used only 4 and the upgrade to 8 probably came about as a concession to the numerical analysis community.

In a lower-cost, lower-speed floating-point unit than the one being described here, the decision as to whether or not to round up could be done in very minimal external logic, since the 74S530 shifter has a "sticky-bit" output which is the OR of all bits shifted out. This output can be cascaded to be valid for an array of 74S530s, like the data outputs.[r7]

The remaining hardware needed for floating-point addition and subtraction, and shown in Figure 2, is relatively minor and has already been discussed, at least indirectly. Figure 5 shows the *left shifter.* The *exponent selector* is basically an exponent-length 2:1 mux, and can be implemented with 3 74S157 (totem-pole) or 74S257 (three-state) muxes. The *exponent corrector* is an exponent-length incrementer/decrementer, and can be implemented with 3 74S382A ALUs connected in a ripple-carry configuration, plus a small amount of external logic. The *sign-generator* is, at least in principle, one exclusive-OR gate.

Fixed-point-to-floating-point conversion is a special case of floating-point addition, and can be performed with only slight additions to the hardware floating-point adder/subtracter unit which we have described. Since the particular fixed constants needed for this algorithm depend on the presumed scale factor of the fixed-point operands to be input into the algorithm, we will consider one typical and particularly important case, where the input operands are 64-bit twos-complement integers. It is a very common choice in more recently-defined computer architectures, for instance the IBM 360/370 and Digital Equipment Corporation PDP-11, for floating-point operands to be sign-magnitude while fixed-point operands are twos-complement. The algorithm to be described here could be implemented as a sequence of instructions, strictly as software, but for economic reasons is likely to be implemented in microcode slightly different from that for floating-point addition and subtraction.

The first step is to render the magnitude of the integer as positive by subtracting it from zero in the 64-bit adder if it is negative, while retaining its original sign as the sign bit of the final result. Next, the characteristic 10000110100, corresponding to an exponent of 52 in decimal, is forced as the characteristic of one operand; the mantissa of that operand is set to the integer absolute value just obtained; the entire second operand is set to zero; and the usual floating-point-addition sequence is followed from that point on, with the minor difference that if the integer had more than 52 significant bits the characteristic of the final result may need to be incremented by more than 1 by the exponent corrector. In this situation, of course, some of the low-end bits of the integer are lost and rounding occurs in the manner already discussed.

Observant readers may have noted by now that the reason this algorithm works is consistent with the fact that operands to be added using the floating-point-addition (or, for that matter, subtraction) algorithm do not actually have to be normalized at the outset. If they are, more accuracy is retained, but the algorithm as it has been described will produce a normalized result in any case. The standard representation for a floating-point zero is 64 zeroes, just as would be used for a 64-bit fixed-point zero. However, an "unnormal zero" with a zero mantissa but some other-than-zero characteristic can also be added or subtracted with like any other floating-point operand, and the result will still be normalized. Some computers, however, provide separate "unnormalized floating add" and "unnormalized floating subtract" operations in which the final normalization step for the usual form of these floating-point operations is suppressed.

Floating-point-to-fixed-point conversion is also a modified form of floating-point addition. In essence, it consists of adding an "unnormal zero" to a floating-point operand, and not normalizing the final result, as in the "unnormalized floating add" operation alluded to above. The important special case here again is where the fixed-point result is to be an integer. If the floating-point operand has a characteristic of at most 10000111111, corresponding to an exponent of 63 in decimal, then 10000111111 is the characteristic of the implied unnormal zero. What then occurs is that the exponent of the operand gets subtracted from 63, and the mantissa of the operand gets shifted right 63-EXP places, which positions the fractional part to be lopped off. The integer part is retained, and placed in a field of zeroes, since it is now a fixed-point datum and has no need for a characteristic. If the sign of the original operand was positive, this integer part with catenated zeroes is the final answer; if not, it is subtracted from zero in the 64-bit adder using fixed-point subtraction to obtain the final answer.

If the original operand was too small to survive this sort of treatment, say of magnitude less than 1, what happens is that shifting it right 63-EXP places shifts it entirely out, since "EXP" in this case is a negative number. So the final answer is zero. Such a number needs to be multiplied by some known scale factor, using floating-point multiplication, before conversion to fixed-point representation is attempted using the above "integerize" algorithm.

Multiplication

Algorithm

The algorithms for multiplication, division, and square root all make use of the same hardware, shown in Figure 8, except that the blocks marked "1Kx8 initial reciprocal" and "pass/complement" are not needed during multiplication. For both multiplication and division, the sign bit of the result is simply the exclusive-OR of the sign bits of the original operands, and for square root the sign of the result is always taken as positive. The characteristic of the product or quotient or root is respectively the sum or difference or half of the characteristics of the original operands, possibly corrected by 1 according to how the result mantissa turned out.

For multiplication, the multiplier and multiplicand mantissas are conceptually divided up into 8-bit groups, beginning at the left, and in a sense extended to 56 bits by the catenation of 4 zeroes on the right. Each 8-bit group of the multiplier needs to be multiplied by each 8-bit group of the multiplicand, so that in principle there are a total of 49 8x8 partial products, as implied by the total outline of the array of Figure 9 including the unnumbered striped areas. However, not all computer architectures require the capability of computing the least-significant half of a double-length product of two long-format floating-point operands. And, if only the most-significant half is to be computed the number of partial products decreases to 34 if the multiplication is basically 56x56^{r1}, and even further to 28 for 52x52 multiplication, since including any additional partial products would accomplish nothing except a tiny improvement in the statistical behavior of the rounding process. For tutorial reasons, the basic description given here of the partial-product generator and the "compressor" (Wallace-tree summation logic) is couched in terms of 56x56 multiplication rather than 52x52 multiplication, with comments as to what hardware changes occur with the slightly-reduced mantissa length. Also, the manner in which the partial products are arranged in Figure 9 differs from the straighforward "rhombus" which one can imagine, if two 56-bit binary numbers were multiplied together on paper and the 56x56 diamond-shaped array of 1s and 0s were divided up into subdiamonds corresponding to each of the 49 partial products, with some of these ignored as before.

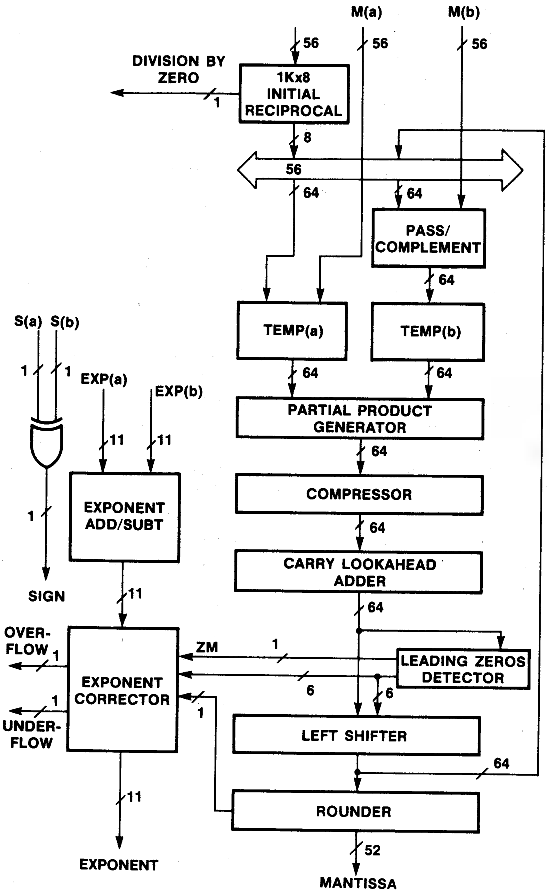


Figure 8. Floating-Point Multiplier/Divider Unit

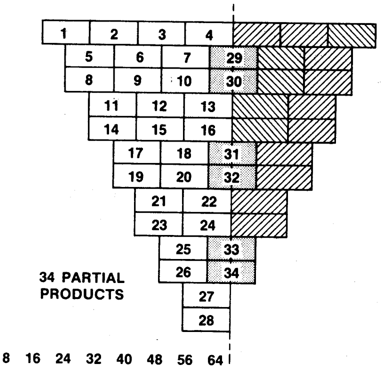


Figure 9. Partial-Product Generator Using 74S558s

In any case, in Figure 9 each numbered rectangle corresponds to one 16-bit partial product available at the parallel outputs of one 74S558 8x8 multiplier, described later. A vertical slice thorugh this array indicates how many bits must be added up in each bit position; the worst-case situation, at the middle of the array is 13 bits. Rather than using a large number of carry-lookahead adders to sum this many bits, a different architecture is employed in which one component sums all the bits which line up with the same bit-position in the answer. This sum can be at most 13, and hence can be represented in 4 bits. For some of the bit-positions towards the left side of the array, there are fewer bits in the corresponding vertical slice and hence the sum for a slice can be represented in 3 or 2 bits. The result at this stage is still an overlapped collection of short binary operands, each in principle beginning at one of the bit-positions of the final answer and extending left. If the same slice-summation strategy is then applied a second time, there can be at most 4 bits in a vertical slice since these short binary operands are at most 4-bit numbers as already mentioned; and hence on this second go-around the result is an overlapped array of 3-bit numbers, since the sum of 4 bits in one bit position is at most 4. In effect, these 3-bit numbers are taken in pairs and summed to produce 4-bit numbers, of at most 12, each of which corresponds to two bit-positions rather than one. Since it is possible to arrange 4-bit numbers staggered by 2 bit-positions into 2 full-length operands, the addition task has now been compressed to the point where the final step can be performed by a 64-bit carry-lookahead adder of the type previously described. The final sum may or may not have to be shifted left one place to normalize it; if it is shifted, the exponent of the product is reduced by one, otherwise the exponent of the product is simply the sum of the exponents. The final step of rounding and lopping off the guard bits proceeds just as previously described for floating-point addition and subtraction. The product exponent is converted back to a characteristic, as before.

Hardware

The partial-product generator is an array of 34 (or 28) 74S558 8x8 Cray multipliers. The 74S558 is a speeded-up successor to the industry-standard 67558; at 60 nsec worst-case for the 16-bit product of two 8-bit numbers, it is more than twice as fast. The 74S558 is a conceptually simple part, although it includes a lot of sheer arithmetic logic. It multiplies two 8x8 binary numbers on a flow-through basis in much the same way that one would multiply them using pencil and paper. It consists of full adders and carry-propagation logic, and does not include any registers or latches. Multipliers of this type are also called "combinatorial," but the internal design uses arithmetic techniques rather than exhaustive Boolean combinations. The 74S558 also includes some capabilities which are not needed in this application: internal rounding controls, arithmetic mode controls, and three-state outputs with an enable/disable control. The arithmetic mode controls allow, in other applications, an independent choice to be made for each of the two operands as to whether it is interpreted as an 8-bit unsigned integer or as a 7-bit twos-complement integer plus a sign bit. Several applications notes$^{r3, r11, r12, r13}$ describe design philosophies using this part and its predecessor the 67558. Figure 10 is a simple but suggestive picture, relating the operation of the part to 8x8 pencil-and-paper binary multiplication.

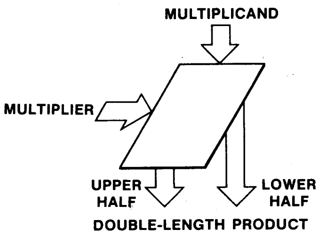


Figure 10. Pencil-and-Paper Analogy to 74S558 Operation

The compressor is a Wallace-tree summation arrangement, for reducing 13 binary operands to be summed down to 2 operands so that a single carry-lookahead adder can do the job. The actual physical components used in the compressor are mostly PROMs of various sizes. Although it is possible to use a single type of Wallace-tree summer everywhere in this type of array, here the choice has been made to optimize computational speed by using different sizes of PROMs operating as differently-organized summers, since the partial-product generator has a "depth" of 1, 3, 5, 7, 9, 11, or 13 bits in a vertical slice at various bit-positions in the product. The groupings of the various PROM sizes required for the first stage of the compressor are shown schematically in Figure 11. There are 7 groups, which are given Roman numerals I, II, ..., VII.

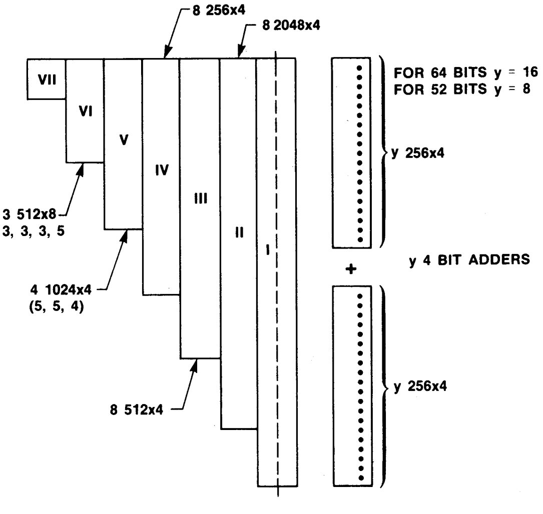


Figure 11. First Level of Compression

Because of entirely pragmatic considerations having to do with the speed of various sizes of available PROMs, group I — where the depth is 13 bits — is broken into two subgroups which use small PROMs, and a 4-bit adder (74S381A or 74S382A) is used for each pair of PROMs, so that the total effect is the same as if a larger PROM had been used. Ideally, a fast 8192x4 PROM would be used for each bit position; the 13 bits from each slice would form the address bits for that PROM, and the 4-bit word read out would range from 0 to 13 according to how many of those address bits were 1s. Since sufficiently-fast 8192x4 PROMs are as yet a little ways in the future, 256x4 PROMs are used which can sum any number of bits up to 8, and operate in 45 nsec. The technique of using two of these per bit slice, and summing their outputs with a 4-bit adder which introduces another delay of 27 nsec worst-case provides a method of summing up a slice of up to 15 bits (16 doesn't occur since the depths are odd) in at most 72 nsec. As noted in Figure 11, a multiplier of up to 64x64 could be constructed using this type of summing arrangement.

For groups II, III, and IV, sufficiently-fast PROMs of the appropriate size are available.r14 Group II uses 2048x4 PROMs, and group IV uses 256x4 PROMs since the 128x4 size is not a standard semiconductor-industry offering. Group V uses 1024x4 PROMs, but now each PROM sums two vertical bit slices instead of one; the PROM address is 10 bits from two 5-deep slices, and the 4-bit number read out can be from 0 to 15 since each 1 bit which turns up in the more-significant slice has a weight of 2 rather than just 1. 1 bits which occur in the less-significant slice have weight 1, just as in groups II, III, and IV. Group VI extends this same approach to sum now for 3 vertical bit slices, using 512x8 PROMs, with weights 4, 2, and 1 for the 3 slices, and possible sum values from 0 to 21. Group VII is a dummy group, since the depth is 1 and no compression is needed.

At the second compression level, instead of 16-bit operands staggered by 8 bits, there are 4-bit operands staggered by 1 bit. Hence as already described in less detail, 256x4 PROMs are used to sum two vertical bit slices at a time through the outputs of the PROMs from the first compression level. Here, the 4-bit number read out can be from 0 to 12. Figure 12 shows the second compression level, with Roman numerals indicating from which group the bits come at the first compression level. Toward the most-significant end, it becomes possible to sum three vertical bit slices with one 256x4 PROM. The notation (2, 2, 3, 4) means that 3 slices are summed — the first can have at most 2 bits with value 1, the second likewise, and the third can have 3, and the total sum can be expressed as a 4-bit number.

The 256x4 PROMs used in the second compression level are numbered from 1 to 25 in Figure 12. At the bottom of the figure, the outputs of these PROMs are shown as they are presented to the carry-lookahead adder. Various symbols are used giving further detail as to where the bits come from: black dots for bits from groups I, IV, and VI of the first compression level, Xs for groups II and V, LS for group III, and 0s for forced zeroes. Xs are also used for the outputs of the dummy group VII, and for the second-level outputs at the bottom.

Bit-slice summers of the type described here are also referred to in the literature as "counters" (although they are utterly unlike the circuits usually called "counters"), "Wallace trees," and "threshold gates," depending on the application context and the bias of the author.

To recap the mantissa multiplication process which has been described: An array of 74S558 multipliers generates 34 16-bit partial products in 13 rows. These 13 rows of bits to be summed are compressed to 4 rows, those 4 rows are compressed to 2 rows, and those 2 rows are added using a conventional binary adder.

Partial products 29-34, corresponding to the lightly gray-shaded rectangles in Figure 9, are not needed for 52x52 multiplication. The number of different PROMs of various sizes in Figure 11 decreases in some cases, but the maximum depth of summation is still 13 bits.

A lower-cost approach to multiplication is suggested in Figure 13. Here, only 7 74S558s are used, so that an 8x56 (or 8x52) partial product is produced. A single full-length carry-lookahead adder suffices to combine the 7 8x8 partial products into an 8x56 partial product, which is actually 64 bits — the least-significant 8 bits, on the right, do not have to go through this adder. A 64-bit accumulator, consisting of a carry-lookahead adder plus a register or else of an array of 2901s or similar registered ALUs, is used to accumulate the 7 8x56 partial products into the final 56x56 answer as they are generated sequentially. Again, if the least-significant half of the double-length product is to be retained, some additional hardware is required.

The remaining hardware in Figure 8 may in fact be the similar hardware shown in Figure 2, if the adder/subtracter unit and multiplier/divider unit are to be partially combined. If it is separate, as is assumed here, some simplifications can be made if the multiplication unit is not also used for division, since if the original multiplier and multiplicand are normalized the result is either normalized or requires at most a one-place left shift.

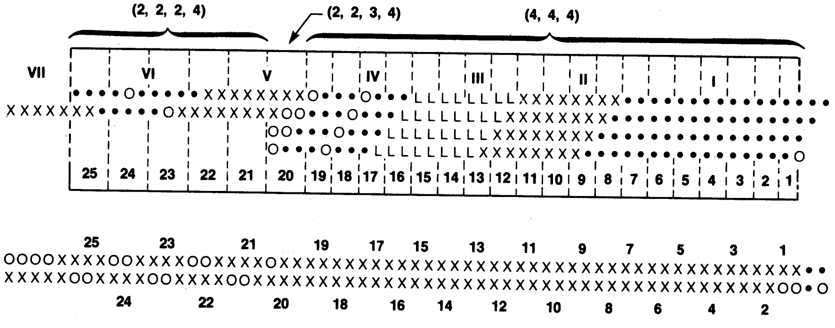


Figure 12. Second Level of Compression

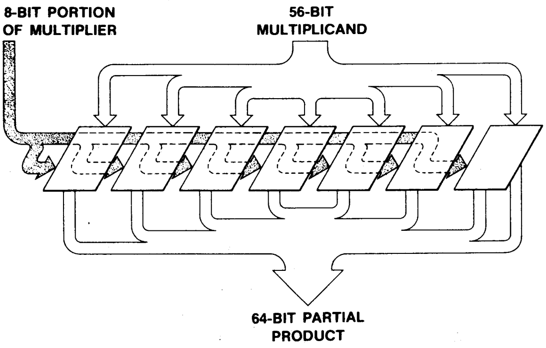


Figure 13. 8x56 Cray Multiplier in Diamond Representation

Division and Square Root

Algorithms

Both the division algorithm and the square root algorithm described here are based on the mathematical procedure known as Newton-Raphson approximation. Given that there is some way to get a starting approximation Q_0, successive approximations are generated according to the formula

$$Q_{n+1} = Q_n - \frac{f(Q_n)}{f'(Q_n)}$$

where f signifies a function of the original operand X and f′ is the calculus derivative of that function. Any standard elementary calculus text contains a proof of this formula, and there are several references available which discuss its application to these algorithms[r5, r15].

These division and square root algorithms are based on using multiplication, rather than subtraction, as the elementary iterated step. The answer they obtain may not agree in the very last bit with that obtained by subtraction-based algorithms, and if for any reason (such as emulation of some other computer) this agreement is essential there is no reason why subtraction-based algorithms cannot be used in the same system as the addition, subtraction, and multiplication algorithms which have been presented here. The reader is referred to discussions of binary restoring, non-restoring, and "non-performing" (more correctly, "conditional entry") division and square root operations in various texts[r16, r17, r18].

The operation actually implemented here, is not (quite) division, but that of obtaining a reciprocal. Once the reciprocal of X, 1/X has been obtained, the multiplication process just described can be used to multiply 1/X by Y to get the quotient Y/X. Consequently, the two functions used in deriving the Newton-Raphson approximation formulas for the algorithms used here are

$$f(X) = 1/X \text{ and } f(X) = \sqrt{X}$$

The Newton-Raphson formulas turn out to be

$$Q_{n+1} = 2Q_n\,(1 - BQ_n) \text{ for the reciprocal}$$

where B is the original divisor whose reciprocal is sought: and

$$Q_{n+1} = \tfrac{1}{2}(Q_n + B/Q_n) \text{ for the square root}$$

of B. This second formula has been familiar to systems programmers for three decades as a means of obtaining square roots when division, but not square root, was provided as a binary arithmetic instruction by the computer designer.

For very-well-behaved functions such as reciprocal and square root, Newton-Raphson approximations have the pleasant property of *quadratic convergence* — if e_n is the error of approximation, then

$$e_{n+1} \le e_n^2$$

which implies that the number of significant bits in the answer *doubles* on each iteration if no other sources of error confuse the issue[r5, r15]. Some care must be taken, however, about the scaling of intermediate results. The Newton-Raphson iteration formula for the reciprocal, above, actually converges to 1/2B rather than to 1/B so that the result is a properly-scaled normalized number.

Returning now to Figure 8, Q_0 is obtained by using the most-significant 10 mantissa bits from B as the address for a 1Kx8 PROM. It is apparent how the iteration formula for division can be microprogrammed using the hardware of Figure 8. If the same hardware is used repeatedly, there must obviously be registers at one or more points, such as in the path from the output of the left shifter back up to the pass/complement unit. Here, it is assumed that registers or latches are provided at the points where boxes labeled "TEMP(a)" and "TEMP(b)" are shown, but that these do not delay the data flow for the straight multiplication algorithm. Because of the quadratic convergence tempered by other sources of error in this implementation, if there are k_n significant bits at the nth iteration, then

$$k_{n-1} = 2k_n - 1$$

and the accuracies of Q_0, Q_1, Q_1, and Q_3 are respectively 8, 15, 29, and 57 bits. The last is, of course, more than is needed for a floating-point data format having a 52-bit or even a 56-bit mantissa. Two multiplications per iteration are required, but that is still just 6 multiplications versus 52 or 56 subtractions if the conventional algorithm were used. The exponent of the quotient, meanwhile, is either the difference of the two original exponents or else the latter quantity plus 1, depending on whether or not the quotient mantissa had to be right-shifted one place to normalize it.

The square-root algorithm can be microprogrammed in much the same way, except that now the iterative step is division, which in turn must be done by iterated multiplication. On the first iteration, the initial trial root and reciprocal can both be obtained in a single table-lookup step. Convergence is just as rapid as for the division algorithm. However, square root is inevitably much slower than division, since it is implemented in terms of division, whereas in the case of subtraction-based square root there is essentially no difference in computing time from that for subtraction-based division.

Conclusion

We have presented algorithms for all usual floating-point operations which can be implemented in a small enough amount of hardware that a system using them might still be considered a minicomputer according to present usage. Although we have postponed making speed estimates for these algorithms to the *APPENDIX,* in order to avoid side issues of computer performance measurement while trying to present clearly how the algorithms work, this computer if constructed would surely be capable of performing 3-4 million floating-point operations per second if evaluated according to the usual instruction "mixes" and otherwise designed according to 1980s standards of minicomputer design engineering. By judicious use of pipelining and concurrency, clever designers can no doubt increase the performance of units using these algorithms to tens of millions of floating-point operations per second in the future.

Acknowledgements

Any tutorial paper on a subject like this one is inevitably the distillation of the knowledge and ingenuity of many people. Here, most of the algorithms presented have appeared first in supercomputers, and the pervasive influence of the design approaches first taken by supercomputer pioneer Seymour Cray has — we hope — shown up throughout. Other major influences on this paper have been two consultants to Monolithic Memories, Daniel Gajski and Dean Hays; several Monolithic Memories customers, notably David Birkner of Computer Design and Applications Inc. and Tor Undheim of Norsk Data A/S; and our boss and colleague Shlomo Waser.

References

r1. "Combinatorial Floating Point Processor as an Integral Part of the Computer," Tor Undheim, *"Electro/80 Professional Program* (spring 1980), paper 14/1. Available from Electronic Conventions, Inc., 999 North Sepulveda Boulevard, El Segundo, CA 90245.

r2. "Shift Matrices: the Missing Teeth in the Number Cruncher," Chuck Hastings, *Wescon/78 Professional Program* (fall 1978), paper 18/3. Available from Electronic Conventions, Inc. (see r1 for address) or from Monolithic Memories, Inc., 1165 East Arques Avenue, Sunnyvale, CA 94086. Note: the 74S530 16x16 shifter was formerly designated as the 67525.

r3. "How to Design Superspeed Cray Multipliers with '558s," Chuck Hastings, applications note available from Monolithic Memories, Inc. (see r2 for address).

r4. "The IBM System/360 Model 91: Floating-Point Execution Unit," S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, *IBM Journal of Research and Development,* Vol. 11 No. 1, January 1967, pages 34-53.

r5. "A Floating-Point Processor Designed for Maximum Speed with Today's TTL LSI and MSI Components," Ehud "Udi" Gordon and Chuck Hastings, applications note available from Monolithic Memories, Inc. (see r2 for address).

r6. *The TTL Data Book for Design Engineers,* Texas Instruments Inc., P.O. Box 225012, Dallas, Texas 75222; second edition. This book has recently gone into its third printing. Page numbers for data sheets for parts which have been used, or mentioned in passing, are: 74S85, 7-57 through 7-64; 74S138, 7-134 through 7-137; 74148, 7-151 through 7-156; 74S151, 7-157 through 7-164; 74S157, 7-181 through 7-187; 74S251, 7-362 through 7-368; 74S257, 7-372 through 7-375; and 74LS348, 7-448 through 7-450.

r7. "Shift Arrays Using the 54/74S530," Dean A. Hays, 11/27/1981, applications note available from Monolithic Memories, Inc. (see r2 for address).

r8. "Doing Your Own Thing in High-Speed Arithmetic," Chuck Hastings, *Proceedings of the Sixth West Coast Computer Faire* (spring 1981), pages 392-410. Available from the Computer Faire, 333 Swett Road, Woodside, CA 94062 or from Monolithic Memories, Inc. (see r2 for address).

r9. *Bipolar Microprocessor Logic and Interface Data Book,* Advanced Micro Devices, Inc., 901 Thompson Place, Sunnyvale, CA 94086. The data sheet for the Am2913 priority encoder is on pages 6-175 through 6-179.

r10. *TTL Applications Handbook,* August 1973, Fairchild Semiconductor, 464 Ellis Street, Mountain View, CA 94042. Pages 4-3 through 4-14 deal with priority encoders.

r11. "Real-Time Processing Gains Ground with Fast Digital Multiplier," Shlomo Waser, *Electronics,* 9/29/1977. Reprints available from Monolithic Memories, Inc. (see r2 for address).

r14. *Bipolar LSI 1982 Databook,* Monolithic Memories, Inc. (see r2 for address). PROMs are in section 3, PALs are in section 6, and arithmetic parts are in sections 10 and 11. Parts which were released to production after August 1981 are not in this databook, and separate self-contained data sheets are available for them from Monolithic Memories.

r15. *Tutorial on Computer Arithmetic,* Engineer degree thesis at Stanford University, Palo Alto, CA 94305, by Shlomo Waser, January 1979. This material is being published as a book, *An Introduction to Arithmetic for Digital System Designers,* by S. Waser and M. J. Flynn, during 1982 by Holt, Rinehart & Winston, N.Y.

r16. *Computer Arithmetic: Principles, Architecture, and Design,* Kai Hwang, John Wiley & Sons, N.Y., 1979.

r17. *The Logic of Computer Arithmetic,* Ivan Flores, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1963.

r18. "High-Speed Arithmetic in Binary Computers," O. L. MacSorley, *Proceedings of the IRE* (now the IEEE), January 1961, pages 67-91.

r19. *Detailed Design of High-Speed Floating-Point Processor Using IBM/370 Format,* Daniel Gajski, applications note available from Monolithic Memories, Inc. (see r2 for address).

Appendix
PAL

FIGURE 2 BLOCK	WORST-CASE LOGIC DELAY (nsec)	PARTS LIST
Shift-Amount Generator (overlaps magnitude comparator	(42)	6 74S382A, 3 74S257
Magnitude comparator	49	17 74S85
Right Shifters (2 required)	20	20 74S530 (10 each)
Carry-Lookahead Adder	64	15 74S381A, 1 74S382A, 5 74S182
Leading-Zeroes Detector	38	9 74S148, 3 74S251
Left Shifter	20	10 74S530
Rounder	61	15 74S381A, 1 74S382A, 5 74S182, 1 PAL16C1
Total:	252	

Table 6. Performance of Floating-Point Addition and Subtraction

FIGURE 8 BLOCK	WORST-CASE LOGIC DELAY (nsec)	PARTS LIST
Partial-Product Generator	60	34 74S558 (28 for 52 bits)
Compressor: First Level Second Level	72 72 45	PROMs: I and IV, 40 256x4 II, 8 2048x4 III, 8 512x4 V, 4 1024x4 VI, 3 512x8 25 256x4
Carry-Lookahead Adder	61	15 74S381A, 1 74S382A, 5 74S182
Subtotal for One Division Step:	238	
Left Shifter	20	10 74S530
Rounder	61	15 74S381A, 1 74S382A, 5 74S182, 1 PAL16C1
Total:	319	175

NOTES
Exponent computations are overlapped with mantissa computations.

Table 7. Performance of Floating-Point Multiplication

FIGURE 8 BLOCK	WORST-CASE LOGIC DELAY (nsec)	PARTS LIST
Initial Reciprocal	35	PROMs: 2 1024x8
Multiply	238, **6 times** (see Table 7)	
Pass/Complement	25	8 PALs
Leading-Zeroes Detector	38	9 74S148, 3 74S251
Left Shifter	20	10 74S530
Total:	1546	32

NOTES
Exponent computations are overlapped with mantissa computations.

Worst-case computation time for square root is essentially 3 times that for division.

Table 8. Performance of Floating-Point Division

PART	DESCRIPTION	SOURCES
74S381A, 74S382A	4-bit ALUs	MMI; FSC
74S182	Carry Bypass	MMI; Others*
74S85	Magnitude Comparator	TI; Others*
74S151, 74S251	8:1 Muxes	TI; Others*
74S157, 74S257	Quad 4:1 Muxes	TI; Others*
74S530	16x16 Shifter	MMI; AMD
74S148, 74S348	8:3 Priority Encoders	MMI; FSC (future)
74S558	8x8 Cray Multiplier	MMI; AMD; FSC (future)
PROMs, in all sizes used here		MMI; AMD, FSC, Harris, NSC, Raytheon, SIG, TI
PALs, in all sizes used here		MMI; AMD, NSC (maybe TI)

NOTES
Abbreviations used in "Sources" are:

AMD: Advanced Micro Devices, Inc.
FSC: Fairchild Semiconductor Corp.
MMI: Monolithic Memories, Inc.
NSC: National Semiconductor Corp.
SIG: SIGnetics Corp.
TI: Texas Instruments, Inc.

* "Others" implies that most or all of AMD, FSC, NSC, SIG, and TI have this part or a direct pin-and-speed-compatible substitute for it in a recent databook.

Table 9. Parts Availability According to Best Current Information

```
PAL16C1                                              PAL DESIGN SPECIFICATION
P7081                                                   VINCENT COLI 03/18/82
ROUNDING-CONTROL LOGIC
MMI SUNNYVALE, CALIFORNIA
D00 D01 D02 D03 D04 D05 D06 D07 D08 GND
D09 D10 D11 D12 TRU COM NC  NC  NC  VCC


TRU =  D12* D11
    +       D11* D10
    +       D11     *D09
    +       D11         * D08
    +       D11             * D07
    +       D11                 * D06
    +       D11                     * D05
    +       D11                         * D04
    +       D11                             * D03
    +       D11                                 * D02
    +       D11                                     * D01
    +       D11                                         * D00


FUNCTION TABLE

D12 D11 D10 D09 D08 D07 D06 D05 D04 D03 D02 D01 D00 COM TRU

;D  D  D  D  D  D  D  D  D  D  D  D  D    ROUND-CONTROL
;12 11 10 09 08 07 06 05 04 03 02 01 00     COM    TRU     COMMENTS
-----------------------------------------------------------------
 X  L  X  X  X  X  X  X  X  X  X  X  X       H      L      TRUNCATE
 L  H  L  L  L  L  L  L  L  L  L  L  L       H      L      TRUNCATE
 H  H  L  L  L  L  L  L  L  L  L  L  L       L      H      ROUND UP
 X  H  H  X  X  X  X  X  X  X  X  X  X       L      H      ROUND UP
 X  H  X  H  X  X  X  X  X  X  X  X  X       L      H      ROUND UP
 X  H  X  X  H  X  X  X  X  X  X  X  X       L      H      ROUND UP
 X  H  X  X  X  H  X  X  X  X  X  X  X       L      H      ROUND UP
 X  H  X  X  X  X  H  X  X  X  X  X  X       L      H      ROUND UP
 X  H  X  X  X  X  X  H  X  X  X  X  X       L      H      ROUND UP
 X  H  X  X  X  X  X  X  H  X  X  X  X       L      H      ROUND UP
 X  H  X  X  X  X  X  X  X  H  X  X  X       L      H      ROUND UP
 X  H  X  X  X  X  X  X  X  X  H  X  X       L      H      ROUND UP
 X  H  X  X  X  X  X  X  X  X  X  H  X       L      H      ROUND UP
 X  H  X  X  X  X  X  X  X  X  X  X  H       L      H      ROUND UP
-----------------------------------------------------------------


DESCRIPTION

THIS PAL16C1 PROVIDES THE ROUNDING-CONTROL LOGIC FOR THE "UNBIASED
ROUNDING" USED IN THE FOLLOWING PAPER ORIGINALLY PRESENTED AT SOUTHCON/82:

BIG, FAST, AND SIMPLE -- ALGORITHMS, ARCHITECTURE, AND
     COMPONENTS FOR HIGH-END SUPERMINIS
     by Ehud Gordon and Chuck Hastings
```

PAL DESIGN SPECIFICATION

```
ROUNDING-CONTROL LOGIC

 1 XXXXXXXXXXXX0XLHXXX1
 2 000000000X0010LHXXX1
 3 000000000X0011HLXXX1
 4 XXXXXXXXXXX11XHLXXX1
 5 XXXXXXXXXX1X1XHLXXX1
 6 XXXXXXXX1XXX1XHLXXX1
 7 XXXXXXX1XXXX1XHLXXX1
 8 XXXXXX1XXXXX1XHLXXX1
 9 XXXXX1XXXXXX1XHLXXX1
10 XXXX1XXXXXXX1XHLXXX1
11 XXX1XXXXXXXX1XHLXXX1
12 XX1XXXXXXXXX1XHLXXX1
13 X1XXXXXXXXXX1XHLXXX1
14 1XXXXXXXXXXX1XHLXXX1

PASS SIMULATION
```

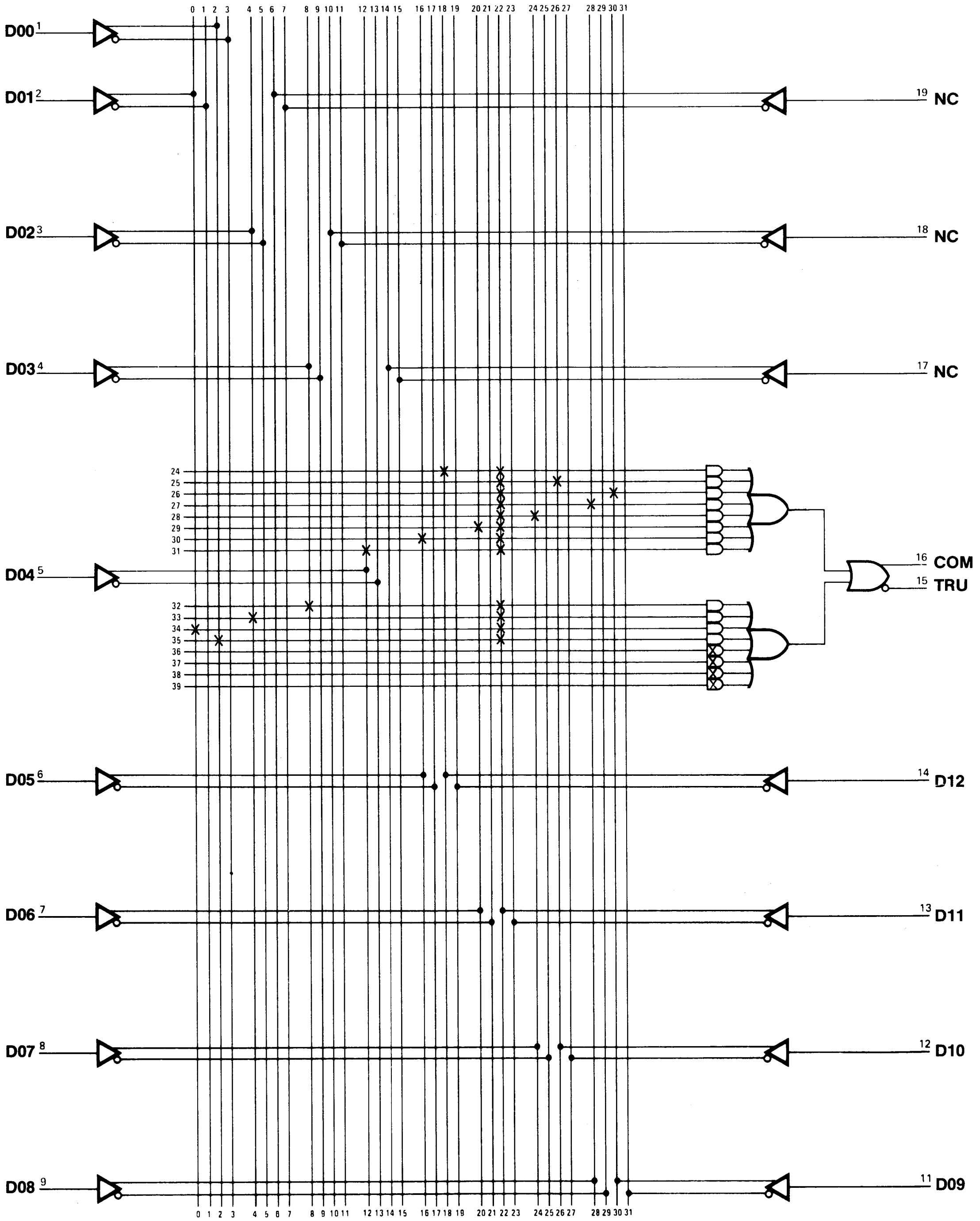
SIMULATION TEST VECTORS

```
ROUNDING-CONTROL LOGIC

              11 1111 1111 2222 2222 2233
   0123 4567 8901 2345 6789 0123 4567 8901


24 ---- ---- ---- ---- --X- --X- ---- ---- D12*D11
25 ---- ---- ---- ---- ---- --X- --X- ---- D11*D10
26 ---- ---- ---- ---- ---- --X- ---- --X- D11*D09
27 ---- ---- ---- ---- ---- --X- ---- X--- D11*D08
28 ---- ---- ---- ---- ---- --X- X--- ---- D11*D07
29 ---- ---- ---- ---- ---- X-X- ---- ---- D11*D06
30 ---- ---- ---- ---- X--- --X- ---- ---- D11*D05
31 ---- ---- ---- X--- ---- --X- ---- ---- D11*D04

32 ---- ---- X--- ---- ---- --X- ---- ---- D11*D03
33 ---- X--- ---- ---- ---- --X- ---- ---- D11*D02
34 X--- ---- ---- ---- ---- --X- ---- ---- D11*D01
35 --X- ---- ---- ---- ---- --X- ---- ---- D11*D00


LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)

NUMBER OF FUSES BLOWN =  360
```

FUSE PLOT

Rounding — Control Logic

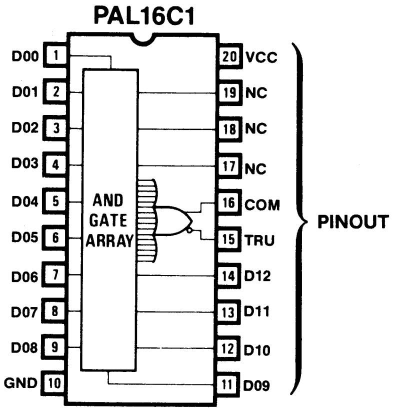
Logic Diagram PAL16C1



```
F F F F F F F F F F F F F F F F F F 7 F F F 7 F F F F F F F F F .
F F F F F F F F F F F F F F F F F F F F F F 7 F F F 7 F F F F F .
F F F F F F F F F F F F F F F F F F F F F F 7 F F F F F F F 7 F .
F F F F F F F F F F F F F F F F F F F F F F 7 F F F F F 7 F F F .
F F F F F F F F F F F F F F F F F F F F F F 7 F 7 F F F F F F F .
F F F F F F F F F F F F F F F F F F F F 7 F 7 F F F F F F F F F .
F F F F F F F F F F F F F F F F 7 F F F F F 7 F F F F F F F F F .
F F F F F F F F F F F F 7 F F F F F F F F F 7 F F F F F F F F F .
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 .
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 .
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 .
1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 .
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 .
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 .
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 .
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 .
```

HEX PROGRAMMING FORMAT

HEX CHECK SUM = EF8



Notes

Four New Ways to go Forth and Multiply

Chuck Hastings

For the last year or so, it has seemed as if every time you turned around Monolithic Memories was announcing *another* new multiplier. These parts generally fall into two categories: 8x8 flow-through Cray multipliers, and bus-oriented sequential multiplier/dividers. Although all of these parts get referred to rather casually as "multipliers," there are major differences between the two general types as to where they fit into designs, how they operate internally, how they are controlled externally, and what they can do and at what speed.

The essential idea of a Cray multiplier, as originally put together by Seymour Cray in the late 1950s with discrete logic at Control Data Corporation, is to wire up an array of full adders in the form of a binary-arithmetic-multiplication pencil-and-paper example. The Monolithic Memories 57/67558, introduced about half a decade ago, was the original *single-chip* Cray multiplier. Many higher-speed versions of this part have since appeared.

In contrast, the Monolithic Memories 'S516 and 'S508 bus-oriented sequential multiplier/dividers are intelligent peripherals for microprocessors, somewhere in between arithmetic sequential circuits and specialized bipolar microprocessors. The 'S516 and 'S508 each can perform any of 28 different multiply and multiply-and-accumulate instructions, plus any of 13 different divide instructions, at bipolar speeds, under the control of an internal state counter.

Four New Ways to go Forth and Multiply

Our Multiplier Population Explosion

For the last year or so, it has seemed as if every time you turned around Monolithic Memories was announcing *another* new multiplier. Want to catch your breath, and find out where each of these fits into the overall scheme of things? Read on.

Actually, there have been *four* new multipliers in all within the last year, plus two which had already been available for several years. In time order of introduction, these are:

Part #	**Description***
57/67558	150-nsec 8x8 Flow-Through Cray Multiplier
57/67558-1	125-nsec 8x8 Flow-Through Cray Multiplier
54/74S508	8-Bit Bus-Oriented Sequential Multiplier/ Divider
54/74S558	60-nsec 8x8 Flow-Through Cray Multiplier
54/74S557	60-nsec 8x8 Flow-Through Cray Multiplier with Transparent Output Latches
54/74S516	16-Bit Bus-Oriented Sequential Multiplier/ Divider

*Times are worst-case times for commercial-temperature-range parts.

You will notice that the above parts fall into two categories: 8x8 flow-through Cray multipliers, and bus-oriented sequential multiplier/dividers. Although all of these parts get referred to rather casually as "multipliers," there are major differences between the two general types; see Table 1 below.

The Cray Multipliers

The essential idea of a Cray multiplier, as originally put together by Seymour Cray in the late 1950s with discrete logic at Control Data Corporation, is to wire up an array of full adders in the form of a binary-arithmetic-multiplication pencil-and-paper example.[3] That is, everywhere that there is a "1" or a "0" in a longhand binary-multiplication example, the Cray type of multiplication uses a full adder. One may visualize a Cray multiplier functionally as a "diamond," as follows:

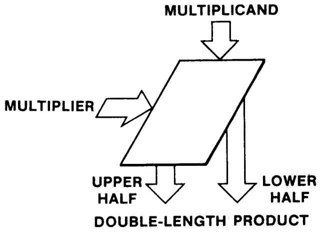


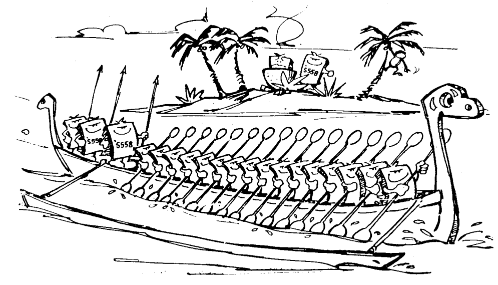
Figure 1. Pencil-and-Paper Analogy to Cray-Multiplier Operation

	8x8 Flow-Through Cray Multiplier	**Bus-Oriented Sequential Multiplier/Divider**
Role in System	*Building-block role*—as many as 34 parts used in one superminicomputer (NORD-500 from Norsk Data[1])	*Co-processor role*—one, or occasionally two, parts used in one microcomputer[2]
Internal Operation	Static arithmetic-logic network; multiplies without being clocked,[3] using eight bits of the multiplier at a time.	State machine; requires clocking to operate; contains edge-triggered registers; sequenced by a state counter; multiplies using two bits of the multiplier at a time[4]
External Control	Controlled by several mode-control input signals	Controlled by sequences of micro-opcodes which come from a microprocessor, a registered PAL, or some other sequential-control device.
Package	40-pin DIP	24-pin DIP
Operations Performed	Can only perform multiplication	Can perform multiplication, division, and multiplication-with-accumulation
Storage Capabilities	Either no storage capabilities (558 types) or optional storage for the double length productivity only (557 types)	Four full-length registers; capable of storing both input operands and the double-length product
Second Sources	Multiple-sourced (AMD, Fairchild, Monolithic Memories)	Sole-sourced; only bipolar *dividers* on the market
Where Used	Initial usage has been in high-end minicomputers, array processors, and signal processors	Initial usage has been in industrial-control microcomputers, digital modems, military avionics, CRT graphic systems, video games, and cartographic analysis systems
Future Prospects	Potential large market today since these parts are now low-cost and multiple-sourced, and should be used in *all* new minicomputer designs!	Potential huge world-wide market for enhancement of microprocessor, bit-slice processor, and microcomputer capabilities, and for small-scale signal processing!

Table 1. A comparison of the two types of Monolithic Memories Multipliers

Our 57/67558, introduced about half a decade ago, was the original *single-chip* Cray multiplier. To achieve what was for that time very high performance for a Schottky-TTL-technology part, the internal design of the 57/67558 also exploited other speed-freak multiplication techniques such as Booth multiplication[4] and Wallace-Tree addition[5]. All of these techniques achieve increased speed through extensive parallelism, and can be used at the system level as well as within LSI components. Subsequently, process improvements made it possible to offer a faster final-test option, the 57/67558-1, which attained a sales-volume level essentially equal to that of the original part.

About three years ago, AMD paid us the sincere compliment of second-sourcing these parts with the 75-nsec 25S558. A few months ago, we returned the compliment with the 60-nsec 54/74S558. All of these '558 parts, and the 70-nsec 54/74F558 announced by Fairchild, are fully-compatible drop-in equivalents except for the variations in logic delay.



"ALL OF THESE TECHNIQUES ACHIEVE INCREASED SPEED THROUGH EXTENSIVE PARELLELISM."

When AMD introduced the 25S558, they introduced along with it the 80-nsec 25S557, a "metal option" of the same basic design with "transparent" output latches to hold the double-length product. "Transparent" means that the latches go away when you don't want them there; a latch-control line like that of the 54/74S373 controls whether these output latches store information, or simply behave as output buffers. Anyway, when we introduced our 54/74S558, we followed it within a few weeks with the 60-nsec 54/74S557, which is a much faster drop-in replacement for AMD's part. And subsequently, Fairchild has announced a 70-nsec 54/74F557.

Because AMD's 'S557 has the output latches implemented in TTL technology *after* the ECL-to-TTL converters, whereas our 'S557 has them implemented in ECL technology *before* the conversion, the latches operate much faster in ours. Our 'S557 is typically only about a nanosecond slower than our 'S558, whereas the logic-delay difference between AMD's two parts is considerably greater. Consequently, our margin of superiority over AMD for the 'S557 is even greater than for the 'S558.

'S557/8 Cray multipliers come in a 40-pin ceramic dual-inline package. The data-bus outputs can sink up to 8 mA I_{OL}. Worst-case power-supply current is 280 mA.

Reference 5 discusses technical approaches to using Cray multipliers in high-performance minicomputers. The 'S558, together with PROMs organized in a "Wallace-tree" configuration, can sail right along at the rate of four 56x56 multiplications every microsecond, on the basis of fixed-point arithmetic with no renormalization. (See table 7 on page 16 of reference 5; the multiplication time is 238 nsec for a "division step," which is a fixed-point multiplication, and 319 nsec for a floating-point multiplication where extra time is required for renormalization and correction of the exponent of the product.) 34 'S558s or 'S557s are required to perform this multiplication if the computer system architecture does not call for the computation of the least-significant half of the double-length product; 49 are required if it does.

"...THE 'S558, TOGETHER WITH PROMS ORGANIZED IN A "WALLACE-TREE" CONFIGURATION, CAN SAIL RIGHT ALONG AT THE RATE OF FOUR 56 x 56 MULTIPLICATIONS EVERY MICROSECOND... "

The "local" architecture of the multiplier section of a digital system can take two rather different forms. A minicomputer[5] which executes an unpredictable mixture of arithmetic and logical instructions one after the other, typically needs to be able to get the complete multiplication over and done with before going on to the next program step—which is probably not another multiplication. An array processor or digital correlator, however, tends to do very regular iterative computations; and the performance of such a system can often be greatly increased by a technique called "pipelining," in which the arithmetic unit consists of stages with registers or latches in between each stage, and partial computational results move from one stage to the next on each clock.

The "flow-through" architecture of the 'S558 works equally well in synchronous or asynchronous pipelined systems, but registers or latches must be provided externally. The 'S557, however, is actually a *superset* of the 'S558, and the added internal-output-latch feature adapts it particularly well to pipelined systems.

"...THE 'FLOW-THROUGH' ARCHITECTURE OF THE 'S558 WORKS EQUALLY WELL IN SYNCHRONOUS OR ASYNCHRONOUS PIPELINED SYSTEMS... "

Even a smaller-scale system can make effective use of these parts. To return to the case of 56x56 multiplication, which corresponds to the word-length needed for multiplying mantissas in several popular floating-point-number formats, an iterative clocked scheme using just seven multipliers, some adders, and an accumulator register can form the entire 112-bit double-length product in just seven multiply/add cycles. A number of mid-range minicomputers today multiply in this manner. The multipliers are configured as suggested by the following block diagram:

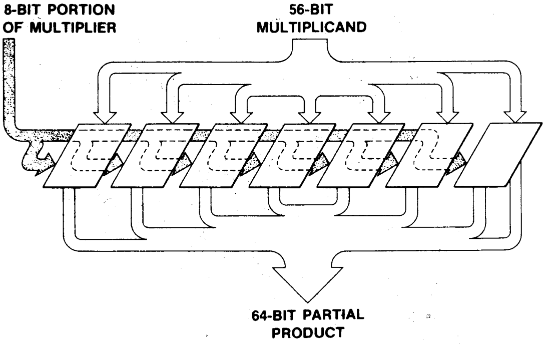


Figure 2. 8x56 Cray Multiplier In Diamond Representation

There is even an occasional 8-bit or 16-bit microprocessor-based system with a need for *very* fast multiplication, where 'S557/8s may get used as microprocessor peripherals[6] Digital-video systems, in particular electronic games, with "vector graphic" capabilities are one example.

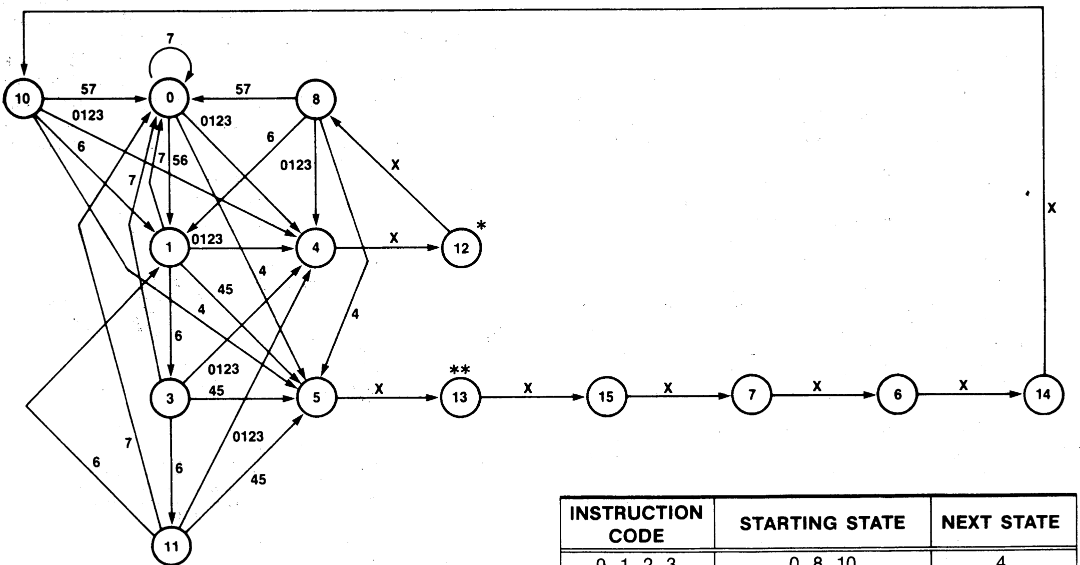
The world of 'S557/8 applications has turned out to include *all* sizes of minicomputers, digital video systems, and signal processors—FFT (Fast Fourier Transform) processors, voice recognition equipment, radar systems, digital correlators and filters, electronic seismographs, brain and body scanners, and so forth. And there are many unexpected offbeat applications, such as real-time data-rescaling circuits in instruments, altogether too numerous to list here. After all, an 'S557/8 can multiply two 8-bit numbers together and output their entire 16-bit product in 60 nsec *worst-case*... less time than it would take a speeding bullet to move the distance equal to the thickness of this piece of paper. How's that for Supermultiplier?

The Multiplier/Dividers

The Monolithic Memories 'S516 and 'S508 are state-of-the-art TTL-compatible intelligent peripherals for microprocessors, somewhere between arithmetic sequential circuits and specialized bipolar microprocessors. The 'S516 and 'S508 each can perform any of 28 different multiply and multiply-and-accumulate instructions, plus any of 13 different divide instructions, at bipolar speeds under the control of an internal state counter. (See Figure 3.) The state counter's sequence is in turn guided by 3-bit instruction codes which are external inputs to the 'S516/ 508. The 'S516 computes with 16-bit binary numbers, and the 'S508 computes with 8-bit binary numbers, as the part numbers none-too-subtly imply.

A 16-bit bidirectional data bus connects the 'S516 with the outside world for bringing in multipliers, multiplicands, dividends, and divisors; and returning products, quotients and remainders. It also has clock (CK) and run/wait ($\overline{\text{GO}}$) inputs, and an overflow indication (OVR) output.

The 'S508 has all of the above inputs and outputs also, except that it has only an 8-bit bidirectional data bus. Since it comes in the same 24-pin package as the 'S516, it obviously has eight more pins available for other purposes. Four of these are used to bring out the internal-state-counter value; one each is used for a completion ($\overline{\text{DONE}}$) status output, an output-enable control (OE) input, and a master-reset ($\overline{\text{MR}}$) control input; and one is not used at all.



*Loop 7 times for multiplication.
**Loop 14 times for fractional division, or 15 times for integer division.

INSTRUCTION CODE	STARTING STATE	NEXT STATE
0, 1, 2, 3	0, 8, 10	4
4	0, 8, 10	5
5	0	1
5, 7	8, 10	0
6	0, 8, 10	1
7	0, 8, 10	0

Figure 3. State-Counter Transition Diagram for the 'S516

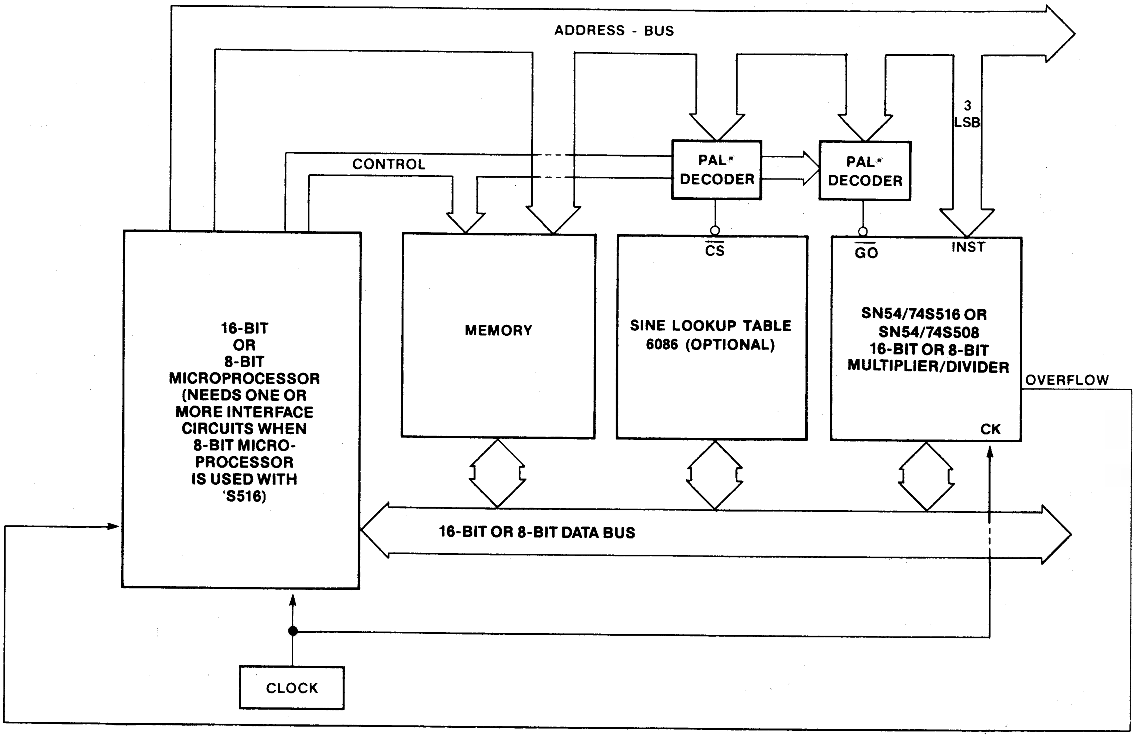


Figure 4. Interfacing the 'S516 to a Microprocessor

A simple, general interfacing scheme can be used to team a 'S516 with any of the currently popular 16-bit microprocessors, or an 'S508 with any 8-bit microprocessor. (See Figure 4.) With a couple extra interface circuits, an 'S516 can also be interfaced to an 8-bit microprocessor. Particularly if the system software is written in a highly-structured language such as PASCAL or FORTH, an 'S516/508 can be retrofitted into an existing system with a large gain in performance and very little impact on either hardware or software—calls to the previous software-implemented one-step-at-a-time multiply and divide subroutines are simply rerouted to substitute a command from the microprocessor to the 'S516/508 to accept an operand and start its operation sequence.

The 'S516 and 'S508 are in fact two different "metal options" of one basic design; the 'S516 has twice as many data bits in each internal register. The 'S516 and 'S508 both have a worst-case clock rate of 6 MHz (commercial) or 5 MHz (military); the typical rate is 8 MHz. The simplest complete twos-complement 16x16 multiplication instruction can be performed in nine clock cycles by an 'S516, or in five by an 'S508, since 2-bits-at-a-time Booth multiplication is used;[4] thus, the worst-case time required by the 'S516 to multiply in this mode is 1.5 μsec for a commercial part, and for an 'S508 it is 833 nsec. On the same basis, 32/16 division can be done in 21 clock cycles, or 3.5 μsec worst-case, by an 'S516; and 16/8 division can be done in 13 clock cycles, or 2.2 μsec worst-case, by an 'S508.

An 'S516/508 can perform either positive or negative multiplication or multiply-accumulation, and many of the instructions provide for "chaining" of successive computations to eliminate extra operand transfers on the bus; these features further enhance the computational speed of the 'S516/508 in particular applications. Arithmetic can be either integer or fractional with respect to positioning of the results.

An 'S516 can powerfully enhance the capabilities of *any* present-day 16-bit or 8-bit microprocessor in a compute-bound application. In fact, it can be used in any digital system where there is a need to multiply and divide on a bus. An 'S508 can likewise enhance the capabilities of any 8-bit microprocessor.

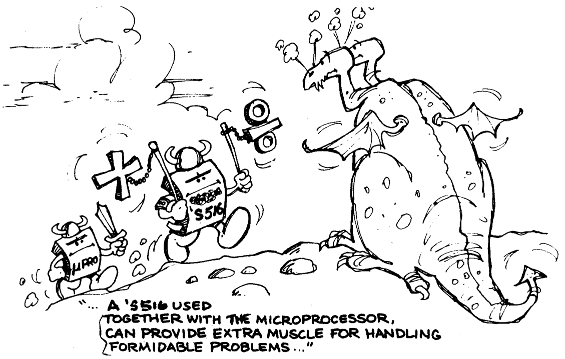
MULTIPLY AND DIVIDE ON A BUS!

The 'S516 comes in an industry-standard 600-mil 24-pin dual-inline package, modified to include an integral aluminum heatsink which does not add appreciably to the package height. It requires only +5V and ground power connections, and draws a worst-case power-supply current of 450mA (commercial) or 500mA (military). Power consumption is greatest at cold temperatures, and decreases substantially as operating temperature increases. The 16 databus inputs require at most 0.25mA input current; the other inputs require at most 1mA. The 16 databus outputs can sink up to 8mA I_{OL}. The 'S508 also fits the above description, except that its worst-case power-supply current is 380mA (commercial) or 400mA (military), and it has only 8 databus inputs and outputs.

In describing applications of these parts, it is difficult to know where to start—they can be used in almost any design where a microprocessor can be used, and you know how many places that is today. So, perhaps a good starting point is to see what uses customers have thought up all by themselves; during the later stages of the development of these parts, many customers took their engineering samples and used these in highly successful designs. One customer even used *two* 'S516s in "pingpong" mode on a single 16-bit bus! So, rather than merely speculating as to what these parts *might* be good for, here's a list of what Monolithic Memories's customers have already *proven* they are good for:

- Real-time control of heavy machinery[7]
- Low-cost, high-performance digital modems
- CRT graphics, including video games
- Military avionics
- Cartographic analysis

As it happens, the above are 'S516 applications, except that digital modem designs have been done with both the 'S516 and the 'S508. One of the 'S516 designs is already in production. In each of these applications, the microprocessor could have coped all right with the computational complexity, albeit at its own less-than-tremendous speed, but a 'S516 used together with the microprocessor can provide extra muscle for handling formidable problems.


'S516
µPRO
"... A 'S516 USED TOGETHER WITH THE MICROPROCESSOR, CAN PROVIDE EXTRA MUSCLE FOR HANDLING FORMIDABLE PROBLEMS..."

Competition? Well, since there are no second sources for the 'S516, and no competitor at present has a similar fast part capable of performing division as well as multiplication, right now the 'S516 has no *direct* competition. Indirectly, there are some competing parts which perform *only* multiplication, and would have to perform division by Newton-Raphson iteration to be usable for any application where division is required. However, the 'S516 is (as far as we know) by far the lowest-priced *bipolar* 16-bit multiplier, and the other microprocessor peripheral chips which can perform division as well as multiplication are relatively-slow MOS devices. In one case, an 8-bit cascadable CMOS part requires a 50% reduction in clock rate to do 16-bit arithmetic. And considerable numerical-analysis and programming sophistication are required to implement Newton-Raphson division with *fixed-point* operands. (It's easier with floating-point operands.) In contrast, the 'S516/508 can be easily interfaced to almost any microprocessor using one or two PALs,® and can perform *either* multiplication or division on command.[2]

The 'S516 is so much faster than the competing MOS chips that it can even take them on for *floating-point* computations (which some of them are designed to do) and *win.* A conference paper[8] describes the design of an 'S516-based S-100-bus card capable of beating an Intel 8087 2:1 on floating-point arithmetic.

Some competing parts, in particular the AMI 2811 and Nippon Electric µPD7720, include an on-board ROM which must be mask-programmed at the factory, which makes life difficult for small companies (or even larger ones) which are trying to get a microprocessor-based product to market quickly. Also, some competing parts require sequencing by external TTL jellybeans.

And, as for using AMD/TRW 64-pin 16x16 Cray multiplier chips as microprocessor peripherals, these cost much more than the 'S516, occupy about three times the circuit-board space, multiply faster, don't divide at all except by Newton-Raphson iteration, and also require one or two "overhead" microprocessor instructions to interface for a given arithmetic operation. From a *system* viewpoint, when this overhead time is reckoned with, these chips provide little actual gain in multiply performance over the 'S516 at lots of extra cost, and an actual loss in divide performance: the 'S516 is *much more cost-effective overall.*

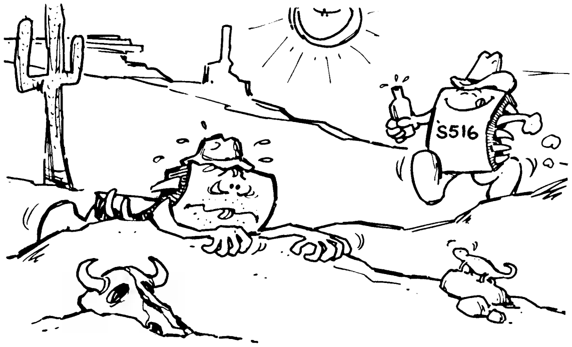
'S516s potentially fit into many, many places in commercial, industrial, and military electronics, particularly into small-scale real-time systems. The part is fast enough to enhance the performance of the 16-bit Motorola 68000, Zilog Z8000, and Intel 8086, as well as that of *any* 8-bit microprocessor. It is also fast enough to considerably improve the multiplication and division performance of 16-bit 2901-based "bit-slice" bipolar microcomputers, which are often used as processors in desktop graphics CRT terminals.

It is worth bringing the 'S516 to the attention of *any* designer who is developing:

- A personal computer or small business computer.
- A word processor, or a more grandiose "office automation system."
- A cruise missile, or any other "smart weapon."
- A digital modem.
- A small-scale speech-processing system. (These are *very* multiplication-intensive. We have one magazine article on the 'S516 in such an application.[9])
- A smart instrument, which does data conversion.
- An industrial control system, particularly one which must do many coordinate transformations.
- An all-digital studio-quality high-fidelity system.
- A cost-reduced computerized medical scanning system.
- A multimicroprocessor system for scientific computations.[10]

If an 'S516/508 is introduced into a system configured around an older microprocessor as a "co-processor" or helpmate for the microprocessor, and the application is arithmetic-intensive, the end effect can be a major upgrading of performance at the system level.[2-7] Consequently, a major reason for designing these parts in is *microprocessor life-cycle enhancement*. In particular, many MOS microprocessors have single-length and double-length add and subtract instructions: but either they have no multiply or divide instructions at all, or else they perform their multiply and divide instructions so slowly as to jeopardize the ability of the entire system to handle its computing load in real time.

So picture, if you will, the entrepreneur or chief engineer of a firm making a successful microprocessor-based widget which has been on the market for a few months, which uses an older 8-bit microprocessor such as a 6800 or 8085 or Z80. Just when his/her sales are really taking off, here comes a new start-up competitor with a similar system, using a Motorola 68000, with added features and faster performance made possible by the 68000's 16-bit word length and multiply/divide capabilities. The 'S516 can, in this instance, serve as a "great equalizer"—it can be retrofitted into the older system as previously described, and provides even higher-speed multiplication and division than the 68000. (Enough so, actually, that there are designers using the 'S516 *with* the 68000.) Thus, the 'S516 can dramatically extend the life cycle of existing microcomputer systems based on microprocessors which either don't have multiplication and division instructions, or perform these operations relatively slowly.



"... THE 'S516 CAN DRAMATICALLY EXTEND THE LIFE CYCLE OF EXISTING MICROCOMPUTER SYSTEMS BASED ON MICROPROCESSORS WHICH EITHER DON'T HAVE MULTIPLICATION AND DIVISION INSTRUCTIONS, OR PERFORM THESE OPERATIONS RELATIVELY SLOWLY..."

'S508s are somewhat easier to control from a logic-design viewpoint than 'S516s, purely because they have more control inputs and outputs. However, the shorter 'S508 word length makes the part naturally fit into smaller-scale systems than those which might use an 'S516. Essentially, the 'S508 is optimized for small-scale systems.

Now that you know what these parts are, can't you think of at least half a dozen prime uses for them right in your own back yard?

References (all available from Monolithic Memories)

1. "Combinatorial Floating Point Processor as an Integral Part of the Computer," Tor Undheim, *Electro/80 Professional Program Session Record,* Session 14 reprint, paper 14/1.
2. "SN54/74S516 Co-Processor Supercharges 68000 Arithmetic," Richard Wm. Blasco, Vincent Coli, Chuck Hastings, and Suneel Rajpal, Monolithic Memories Application Note AN-114.
3. "How to Design Superspeed Cray Multipliers with '558s," Chuck Hastings, pages 20-22, *Structured Logic,* July 1981. A slightly rewritten form of this article has since been reprinted as part of the new SN54/74S557/8 data sheet.
4. "Doing Your Own Thing in High-Speed Digital Arithmetic," Chuck Hastings, Monolithic Memories Conference Proceedings reprint CP-102.
5. "Big, Fast, and Simple—Algorithms, Architecture, and Components for High-End Superminis," Ehud Gordon and Chuck Hastings, Monolithic Memories Application Note AN-111.
6. "An 8x8 Multiplier and 8-bit µp Perform 16x16 Bit Multiplication," Shai Mor, *EDN,* November 5, 1979.
7. "The Design and Application of a High-Speed Multiply/Divide Board for the STD Bus," Michael Linse, Gary Oliver, Kirk Bailey, and Michael Alan Baxter, Monolithic Memories Application Note AN-115.
8. "Minimum Chip-Count Number Cruncher Uses Bipolar Co-Processor," C. Hastings, E. Gordon, and R. Blasco, *Wescon/81 Professional Program Record,* Session 3 reprint, paper 3/1.
9. "Medium-speed Multipliers Trim Cost, Shrink Bandwidth in Speed Transmission," Shlomo Waser and Allen Peterson, *Electronic Design,* February 1, 1979; pages 58-65.
10. "A Synchronous Multi-Microprocessor System for Implementing Digital Signal Processing Algorithms," T. P. Barnwell, III and C. J. M. Hodges, *Southcon/82 Professional Program Session Record,* Session 21 reprint, paper 21/4.

Enhancing 8086 Arithmetic Using The SN54/74S516 Multiplier/Divider

Jerry Greiner, Frank Lee, and Suneel Rajpal

A serious limitation in most microcomputers is arithmetic computation at high speed, especially when multiplications or divisions are required. With minimal interface and programming overhead, the operations can be performed at very high speed by the SN54/74S516 multiplier. This paper describes how a PAL is used to interface the SN54/74S516 to the INTEL 8086 Microprocessor.

Enhancing 8086 Arithmetic Using The SN54/74S516 Multiplier/Divider

Jerry Greiner, Frank Lee, and Suneel Rajpal

Introduction

A Specialized LSI multiplier/divider, the SN54/74S516 can be used together with MOS microprocessors to significantly improve its arithmetic throughput. A serious limitation in most microcomputers however, is arithmetic computation at high speed, especially when multiplications or divisions are required. With minimal interface and programming overhead, multiplication and division are performed at high speed by the 74S516.

This application note describes the interface of the 74S516 to the Intel 8086. An evaluation of the performance of the Intel 8086 with and without the 74S516 for typical applications are tabulated.

Description of the S516

The 'S516 is a bus-organized 16x16 multiplier/divider: its block diagram is given in Figure 1. There are 28 different multiply options, including positive and negative multiply, positive and negative multiply-accumulation, multiplication by a constant, and both single- and double- length addition in conjunction with multiplication. 13 different divide operations allow single- or double- length division, division of a previously generated result, division by a constant, and a continued division by a remainder or quotient. The full instruction set is presented in Table 1.

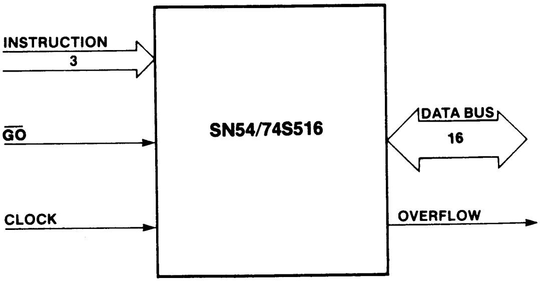


Figure 1. Block Diagram of 'S516

The 74S516 is a time-sequenced device requiring clocks; it loads operands from, and presents results to, a bidirectional 16-bit bus. Loading of the operands, reading of the results and sequencing of the device is controlled by a 3-bit instruction. The operands and results can be either integer or fractions; results may be rounded if required, and an overflow output indicates whenever a result is outside the normally accepted number range.

Detailed operation and interface requirements for the S516 are given in its data sheet. This section will cover some of the more relevant items from the data sheet.

The pin configuration of 'S516 is given in Figure 2. The 'S516 contains four 16-bit working registers. Y is the multiplier register; X is the multiplicand/divisor register; W is the least-significant half of a double-length accumulator. Z is the most significant half of this same accumulator. The accumulator is used to hold the resulting product for multiply operations. The accumulator can also hold the dividend for a 32-bit by 16-bit division. The Z and W registers also hold the quotient and remainder respectively for divide operations. Operands are loaded into the working registers in time sequence at each clock period, under the control of a sequencer. The chip-activation signal $\overline{GO}$ must be LOW during the instruction loading process and during the READ operation. If $\overline{GO}$ is held HIGH, the 'S516 holds its outputs in their high-impedance states, so that other devices attached to the bus may drive it. In this condition, the device completes instruction that have previously been initiated but does not respond to any new codes on its instruction inputs. To read results onto the bidirectional bus, instruction code 7 is specified in addition to the $\overline{GO}$ signal being held LOW. Since there is a double-length accumulator Z, W, reading takes two cycles. First, register Z is read. If code 7 is still present, the W register is placed on the bus at the next cycle.

The 'S516 has no direct master reset input. The initialization of the 'S516 can be performed by continually presenting instruction code 7 for 21 clock cycles.

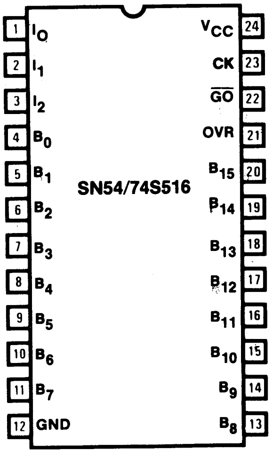


Figure 2. Pin Configuration of 'S516

INSTRUCTION SEQUENCE	OPERATION	CLOCK CYCLES
	ARITHMETIC OPERATIONS	
0	X1 · Y	9
1	-X1 · Y	9
2	X1 · Y + K_Z, K_W	9
3	-X1 · Y + K_Z, K_W	9
4	K_Z, K_W/X1	21
5/6 0	X · Y	10
5/6 1	-X · Y	10
5/6 2	X · Y + K_Z, K_W	10
5/6 3	-X · Y + K_Z, K_W	10
5/6 4	K_W/X	22
5/6 5	K_Z/X	22
5/6 6 0	X · Y + Z	11
5/6 6 1	-X · Y + Z	11
5/6 6 2	X · Y + $K_Z \cdot 2^{-16}$	11
5/6 6 3	-X · Y + $K_Z \cdot 2^{-16}$	11
5/6 6 4	Z, W/X	23
5/6 6 5	Z/X	23
5/6 6 6 0	X · Y + Z, W	12
5/6 6 6 1	-X · Y + Z, W	12
5/6 6 6 2	X · Y + W_{sign}	12
5/6 6 6 3	-X · Y + W_{sign}	12
5/6 6 6 4	W/X	24
5/6 6 6 5	W_{sign}/X	24
5/6 6 6 6	(See Note 9 below.)	—
5/6 6 6 7	Load X, Load Z, Load W, Clear Z	4
5/6 6 7	Load X, Load Z, Read Z	3
	READING OPERATIONS	
7	Read Z	1
7 7	Read Z, W	2
7 7 7	Read Z, W, Z	3
7 7 7 7	Read Z, W, Z, W	4
5 7	Round, then Read Z	2
5 7 7	Round, then Read Z, W	3

NOTES:

1. X,Y are input multiplier and multiplicand.
2. X1 is the previous contents of the first rank of the X register (either the old X or a new X).
3. Fractional or integer arithmetic is specified by having the next-to-the-last operand loaded using a 5 or 6 instruction respectively. All rows beginning with "5/6" in effect represent *two* instructions. 5 does fractional arithmetic and 6 does integer arithmetic.
4. Z, W is a double-precision number. Z is the most significant half. Z, W represents addend upon input, and product (or accumulated sum) after multiplication.
5. K_Z, K_W represents previous accumulator contents. K_Z is the most-significant half.
6. W_{sign} is a single-length signed number, with sign extension.
7. Maximum clock cycle = 167 ns for an 6-MHz clock.
8. If n instruction codes are shown at the left under "instruction sequences," the number of clock cycles at the right is n+8 for multiplication and n+20 for division.
9. The code "5/6 6 6 6" represents an incomplete operation since it leaves the 'S516 in state 1 rather than in state 0, 8, or 10.

Table 1. 'S516 Instruction Set (Partial List)

After the 'S516 is initialized, that is it is in state 0, one can start loading the operands. To understand how the 'S516 works it is important to realize that the instructions are inputs to a state counter as shown in Figure 3. The state counter drives a PLA circuit which in turn controls the other elements in the 'S516. The state transition diagram is given in Figure 4. The seemingly complex state transition diagram is simplified by the following: for example, a 16-bit by 16-bit multiply operation can be performed in 10 clock cycles as shown in Table 2. Out of these,two are used to load the operands and eight to actually perform the multiply routine. After that the 'S516 is in state 8 and the most significant 16 bits of the product can be read from the Z register. After that, if instruction 7 is presented the 'S516 moves to state 0 and the W register is read. The states of the 'S516 for this operation are 0, 1, 4, 12, 12, 12, 12, 12, 12, 12, 8, 0, for the appropriate instruction. Note that in states 0, 1, 3, 8, 10, 11 you require $\overline{GO}$ to enable the clock to the 'S516. Also the 'S516 moves from state 4 to 12 irrespective of the GO and instruction codes, and the same holds when it is in state 12. For states 4 and 12, the clock transition from LOW to HIGH cause state transitions.

An emulation of the 'S516 state counter can be easily built using a PAL. A detailed PAL specification is provided in (1).

The 'S516 data lines may reside directly on the 8086 bus, if loading by other devices is not excessive. The 'S516 data outputs can sink 8 mA. The data inputs use PNP devices equivalent of 1 to 1/2 of an LS-TTL load. The other inputs are equivalent to 1/2 of a standard TTL load.

GO	0	0	X	X	0	0
INPUT INSTRUCTION	6	0	X	X	7	7
S516 STATE	0	1	4	12*	8	0
BUS ACTIVITY	X INPUT	Y INPUT	—	—	Z OUTPUT	W OUTPUT

* The 'S516 remains in state 12 for 7 cycles.

Table 2. The Instruction Sequence for a Multiply Operation

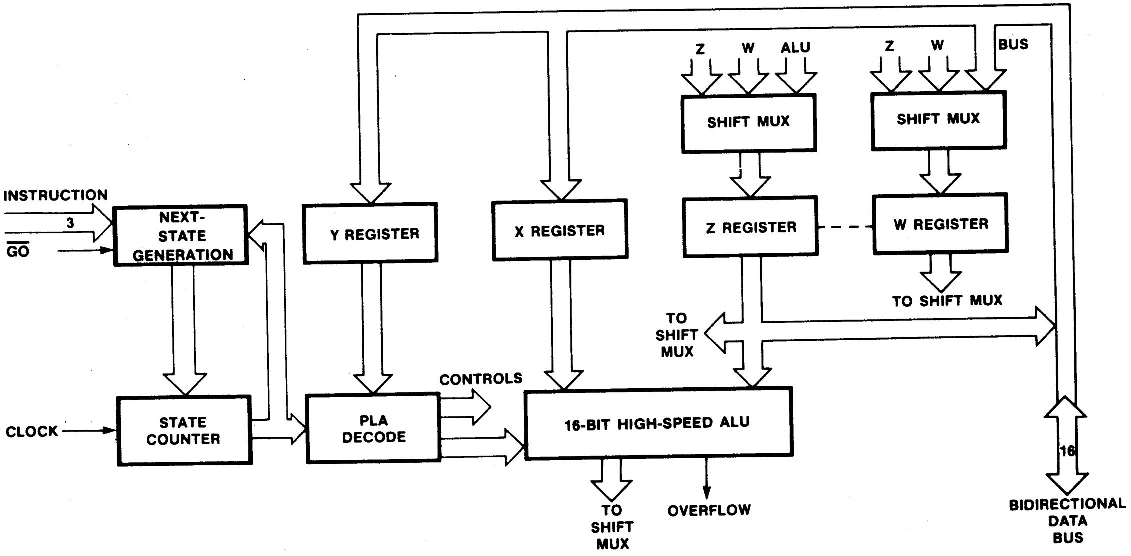
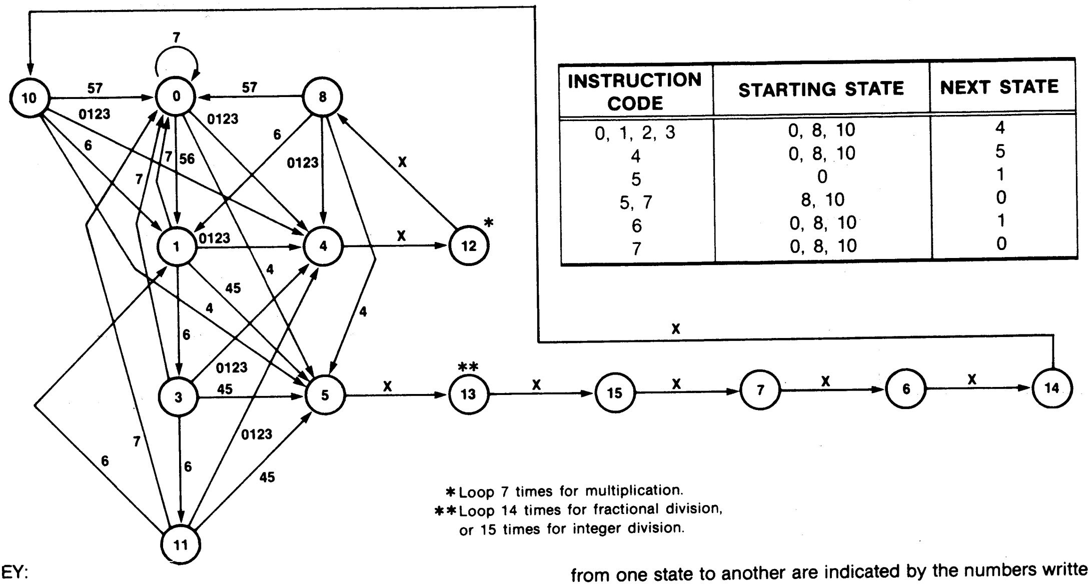


Figure 3. Internal Architecture of the 'S516



INSTRUCTION CODE	STARTING STATE	NEXT STATE
0, 1, 2, 3	0, 8, 10	4
4	0, 8, 10	5
5	0	1
5, 7	8, 10	0
6	0, 8, 10	1
7	0, 8, 10	0

*Loop 7 times for multiplication.
**Loop 14 times for fractional division, or 15 times for integer division.

KEY:

The numbers inside the circles indicate the *state* of the 'S516 multiplier/divider. These states are represented by a four-bit state counter, where A is the least-significant bit of this state counter and D is the most-significant bit. (These four bits are not available externally on the 'S516.)

The next state of the 'S516 is a function of the present state and the instruction lines. For example if the 'S516 is at state 0 and the instruction is 0, 1, 2, or 3, then the next state is state 4 (multiply instruction); if the instruction is 4, the next state is state 5 (divide instruction); and so forth. The instructions which take the 'S516 from one state to another are indicated by the numbers written next to the state-transition path lines. "0123," for instance, implies that *any* of instructions 0, 1, 2, or 3 will take the 'S516 along the path marked "0123."

"X" next to a path implies that the path will be followed regardless of the value of the instruction inputs at that time. In other words, for the purpose of state transitions, X means "don't care." There are cases, however, where the particular instruction used may affect when the contents of the registers are available on the bus — see Figures 9 and 10 of the 'S516 data sheet for contrasting examples of how this effect operates.

Figure 4. Transition Diagram for the 'S516 Multiplier/Divider

Hardware Interface

The Intel 8086 is an advanced, high-speed, 16-bit microprocessor. It has options that operate at 8 MHz, but this design uses a 5 MHz 8086. For a complete specification of Intel 8086, please refer to its data sheet.

The interface of the Intel 8086 to the 'S516 is given in Figure 5. The three instruction lines of the 74S516 are assigned to the three least significant bits of the address bus, while the two most significant address bits are decoded to determine if the 'S516 is selected. A PAL is used to decode the select signal and to provide properly timed $\overline{GO}$ signal and clock signals to the 'S516. With this structure, a simple MOVE instruction by the 8086 can be transformed into any of the multiply/divide routines of the 'S516

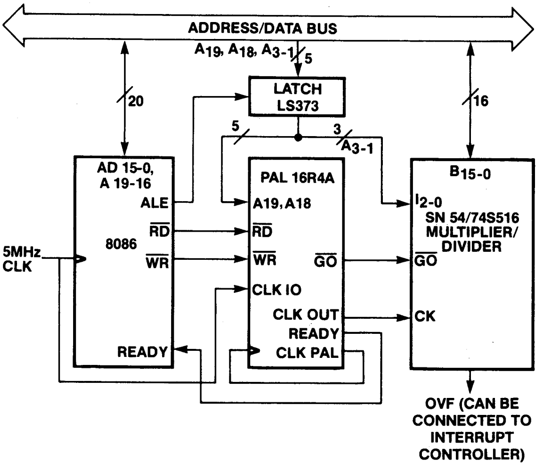


Figure 5. The Interface Diagram for the Intel 8086 to the 'S516

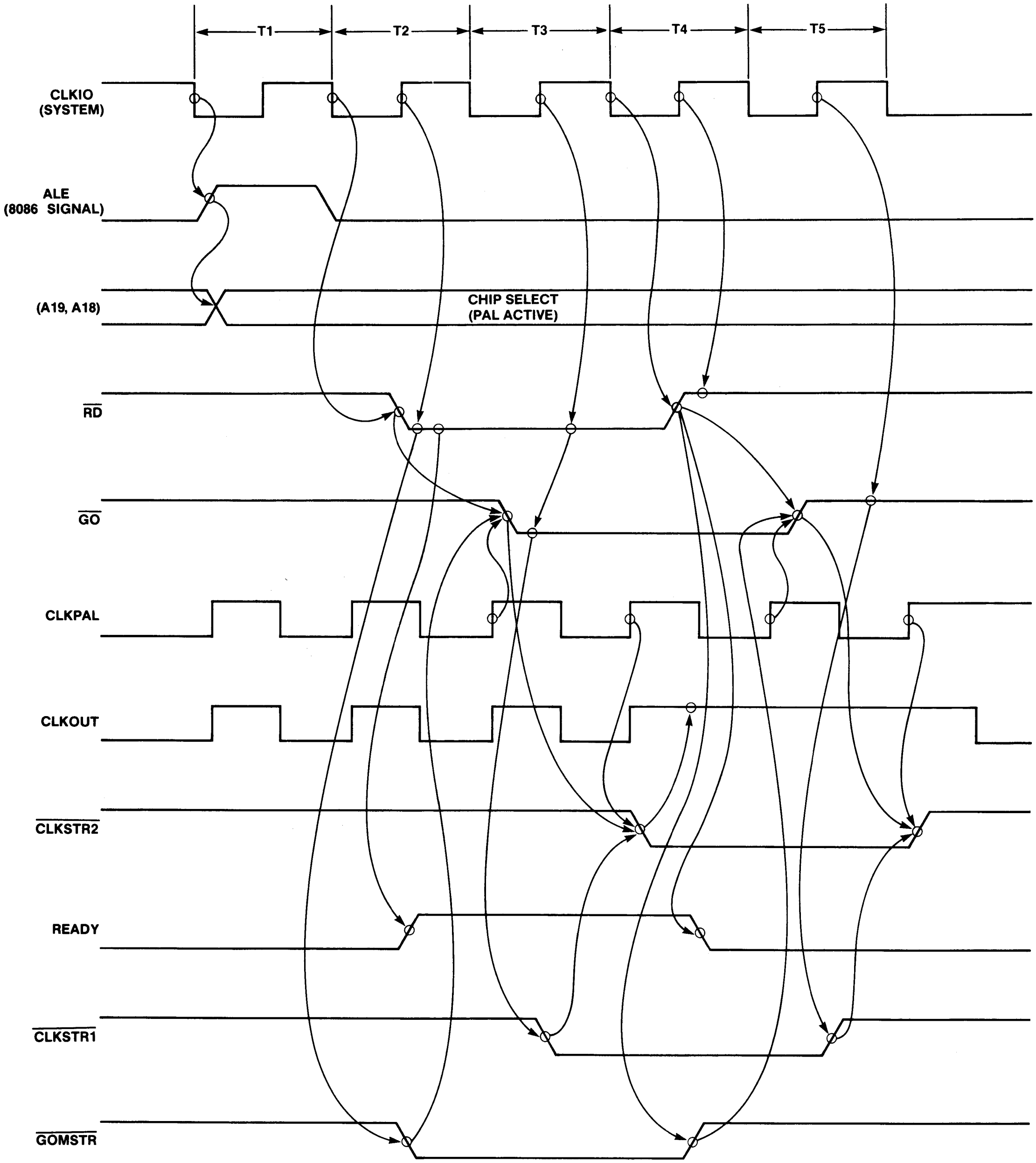


Figure 6. Timing for Read Cycle

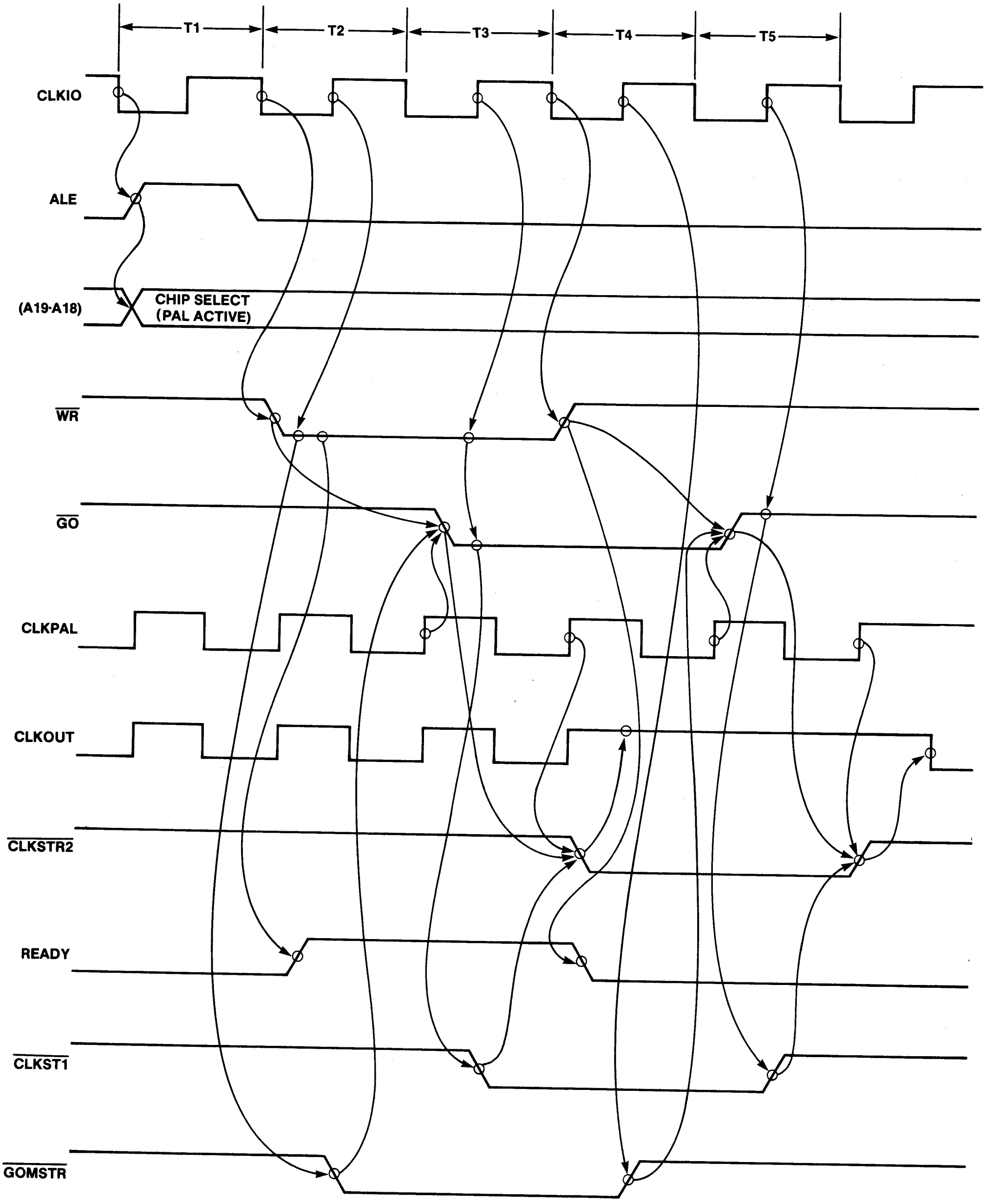


Figure 7. Timing for Write Cycle

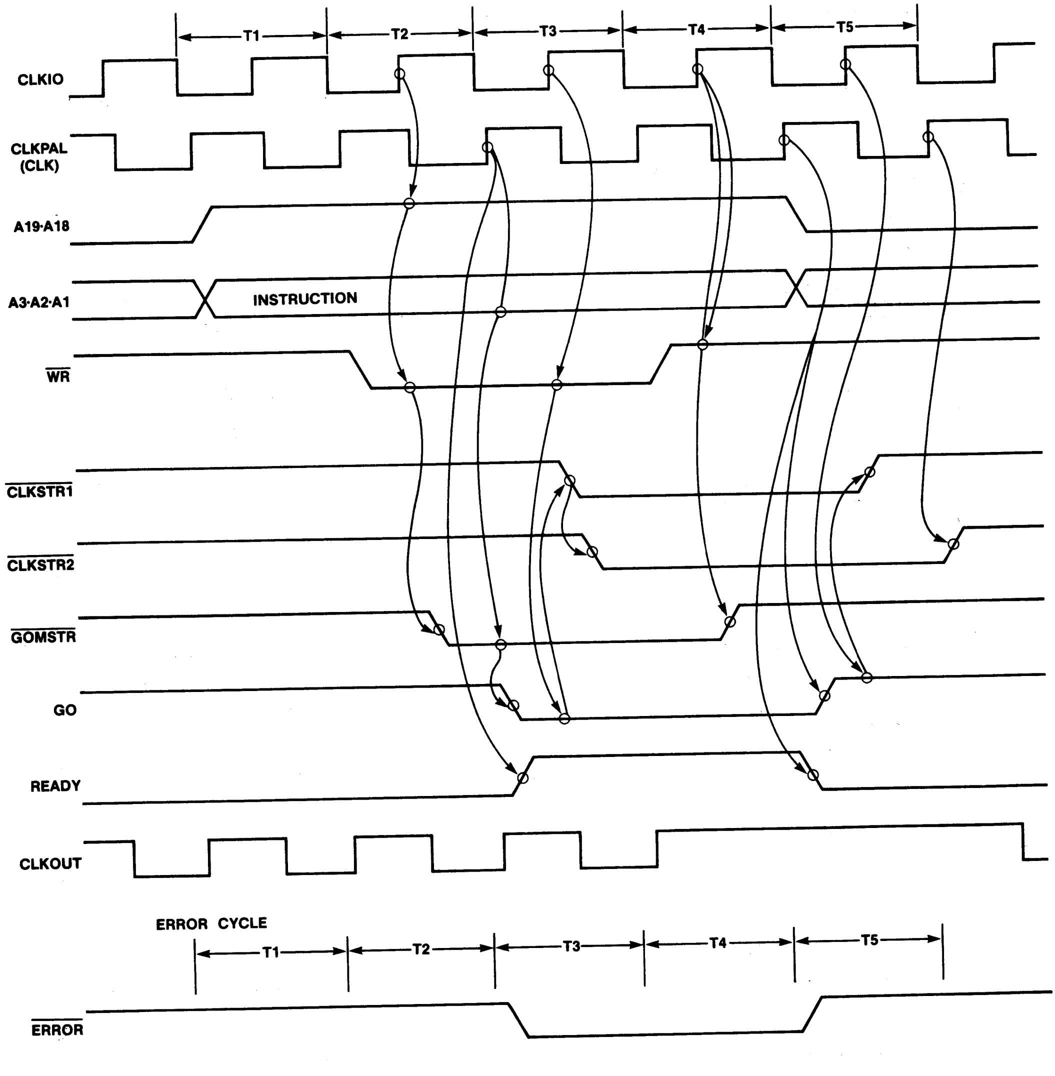


Figure 8. Read and Write Cycle

The following design is based on the 8086 in minimum mode at 5 MHz clock frequency. The objective is to use information on the 8086 address bus and bus control signals to derive the controls for the 'S516 with proper timing. The 8086 bus transfer takes a minimum of four clock cycles. The instruction codes corresponds to the three LSBs of the address bus when the 'S516 is selected. Address lines A19 and A18 are decoded to memory map the 'S516. GO should be properly timed to load instructions and data onto the bus. Considering the timing requirements of both the 8086 and the 'S516, the timing relationships for the interface are derived and shown in Figures 6, 7 and 8. The basic constraints that have to be met is that GO and instruction codes must settle before the rising edge of CK, and they have a minimum hold time of 30 ns. To accomodate the above, the Clock is stretched high (CLKOUT = HI) until GO is disabled.

Interface Design Detail

The two most significant address lines from the 8086 must be asserted to enable the PAL, ie. (A19 = A18 = HI). A3, A2, and A1 are used for instruction I2, I1, and I0 respectively. A0 is not used as the 8086 addresses the 'S516 on a word boundary rather than a byte boundary. The system clock is used as an input (CLKIO) to the PAL. From this clock, two other clock signals are generated. CLKOUT is an inversion of CLKIO and is used to drive the 'S516. It must be inverted to meet the stringent timing constraints of the interface. CLKPAL is also an inversion of CLKIO and is fed back to pin 1 of the PAL and used to drive the registered outputs of the PAL. Again, inversion of the system clock is necessary for timing. Read ($\overline{\text{RD}}$) and Write ($\overline{\text{WR}}$) are signals from the 8086 which are used to generate GO for the 'S516.

To generate a properly timed $\overline{\text{GO}}$ signal for the 'S516, the PAL uses the master/slave principal. Go Master is asserted ($\overline{\text{GOMSTR}}$ = LOW) which in turn asserts Go Slave ($\overline{\text{GO}}$ = LOW). This signal ($\overline{\text{GO}}$) is then sent to the 'S516 to perform the desired operation.

During a Read ($\overline{\text{RD}}$) operation, data is clocked out of the 'S516 in one cycle; however, it takes two cycles for the 8086 to latch in the data. To accomplish this, the 'S516 clock is stretched high (CLKOUT = HI) for two clock cycles. Again, the master/slave principal is used to provide a properly timed clock stretch. Clock Stretch Master is asserted ($\overline{\text{CLKSTR1}}$ = LOW) which in turn asserts Clock Stretch Slave ($\overline{\text{CLKSTR2}}$ = LOW) which holds CLKOUT = HI for the desired interval.

Ready (READY = HI) is sent from the PAL back to the 8086 to acknowledge Read or Write. It should be noted that $\overline{\text{CLKSTR1}}$, CLKSTR2, and $\overline{\text{GOMSTR}}$ are signals used internally by the PAL and are not connected to any external circuitry. Also, CLKPAL must be externally connected back to pin 1 of the PAL. CLKOUT and $\overline{\text{GO}}$ are connected to Clock and $\overline{\text{GO}}$ of the 'S516, respectively. The error flag ($\overline{\text{ERROR}}$) will be asserted if there is an illegal read or write. An illegal read occurs when $\overline{\text{RD}}$ = 0 and A3, A2, A1 ≠ 111. An illegal write occurs when $\overline{\text{WR}}$ = 0 and A3, A2, A1 = 111.

Analysis of Computation Speed

The instruction execution time of a 16-bit multiplication in 8086 is about 128-154 clock cycles, compared to 10 clock cycles for 'S516. This gives 'S516 a speed advantage of 12.8-15.4:1. The raw speed ratio for a 16-bit division is about 6.0-7.1:1. The 'S516 speed advantage decreases, however, when I/O time is considered, as illustrated in the following examples. The following comparisons are made with the assumption that both 8086 and 'S516 are clocked at the same rate.

Programming is simplified by first assigning mnemonics to the absolute addresses for the 'S516 instructions:

```
INSTR0  EQU  FFFF0H;
INSTR1  EQU  FFFF2H;
INSTR2  EQU  FFFF4H;
INSTR3  EQU  FFFF6H;
INSTR4  EQU  FFFF8H;
INSTR5  EQU  FFFFAH;
INSTR6  EQU  FFFFCH;
INSTR7  EQU  FFFFEH.
```

First, a look at a simple multiply routine; the multiplier is stored in AX, and multiplicand in DX, and the result is stored back in these registers:

```
  8086 - 'S516
MOV  INSTR6, AX      ;  load instruction 6, X
MOV  INSTR0, DX      ;  load instruction 0, Y
MOV  DX, INSTR7      ;  read most significant word of result
MOV  AX, INSTR7      ;  read least significant word of result
  total cycles = 53 cycles.

  8086 only
IMUL DX                 ;  multiply
  total cycles = 128-154 cycles.
```

In this case, the speed advantage is 2.4-2.9:1. With the 8086—'S516 system, however, 8086 is fully occupied (no wait cycle), and the 10 clock cycles for 'S516 multiply is fully imbedded in the 8086 cycles. Therefore, a faster multiplier could not improve the throughput.

The following codes are for a 16-bit divide routine; the divisor is in AX, and dividend in DX:

```
  8086 - 'S516
MOV  INSTR6,  CX    ;  load instruction 6, X
MOV  INSTR6,  DX    ;  load instruction 6, Z
MOV  INSTR5,  AX    ;  load instruction 5, W
NOOP                ;
NOOP                ;
NOOP                ;
NOOP                ;
NOOP                ;
MOV  AX,  INSTR7    ;  read quotient
MOV  DX,  INSTR7    ;  read remainder
  total cycles = 81 cycles.

  8086 only
IDIV  CX            ;  divide
  total cycles = 165-184 cycles.
```

The 'S516 provides a 2.0-2.3 times speed improvement. There are 15 wait cycles for 8086. These wait cycles can be utilized for fetching operand from memory if a chained operation is required, as illustrated in the next example.

The above two cases are worst-case examples in terms of speed improvement. More complex operations will show further improvement, as the read/write overhead is reduced relative to the total computation time.

In the last example, we will look at using the 8086 — 'S516 for computing sum-of-products:

Xi*Yi

For this example, we set n = 4. X1 and Y1 are preloaded in AX and BX.

```
  8086 - 'S516
MOV  INSTR6, AX     ;  load instruction 6, X1
MOV  INSTR0, BX     ;  load instruction 0, Y1
MOV  AX, X2         ;  fetch X2
MOV  BX, Y2         ;  fetch Y2
MOV  INSTR6, AX     ;  load instruction 6, X2
MOV  INSTR2, BX     ;  load instruction 2, Y2
MOV  AX, X3         ;  fetch X3
MOV  BX, Y3         ;  fetch Y3
MOV  INSTR6, AX     ;  load instruction 6, X3
MOV  INSTR2, BX     ;  load instruction 2, X3
MOV  AX, X4         ;  fetch X4
MOV  BX, Y4         ;  fetch Y4
MOV  INSTR6, AX     ;  load instruction 6, X4
MOV  INSTR2, BX     ;  load instruction 2, Y4
MOV  AX, INSTR7     ;  read most significant word of result
MOV  DX, INSTR7     ;  read least significant word of result
  total cycles = 212 cycles.
```

```
  8086 only
IMUL BX             ;  multiply X1, Y1
MOV  BX, DX         ;  load result in register BX
MOV  CX, AX         ;  load result in register CX
MOV  AX, X2         ;  load X2
IMUL Y2             ;  multiply X2, Y2
ADD  CX, AX         ;  sum result
ADC  BX, DX         ;  extension
MOV  AX, X3         ;  load X3
IMUL Y3             ;  multiply X3, Y3
ADD  CX, AX         ;  sum result
ADC  BX, DX         ;  extension
MOV  AX, X4         ;  load X4
IMUL Y4             ;  multiply X4, Y4
ADD  AX, CX         ;  sum result
ADC  DX, BX         ;  extension
  total cycles = 612-716 cycles.
```

This example shows over 2.9-3.4:1 speed improvement. The results are tabulated in Table 3.

OPERATION	CYCLES	
	8086	8086+ S516
MULTIPLY (16x16)	154	53
$\sum_{i=1}^{n} a_i x_i$ (n = 4)	776	212
DIVIDE (16/16)	184	81

Table 3. Performance Comparison of the PROM 'S516

Conclusion

The 8086 and 'S516 combination has been demonstrated to be practical and effective. This opens up numerous applications for the 8086 such as signal processing, cartographic analysis, industrial controls where previously it may have not been used, due to lower multiply and divide throughputs. The other features of the 'S516, low cost in terms of dollars and real estate and availability make it a very feasible solution to enhancing microprocessor arithmetic.

```
PAL16R4A                                                  PAL DESIGN SPECIFICATION
8086/516                                             JERRY GREINER/FRANK LEE 1/31/83
8086 CPU/74S516 MULTIPLIER INTERFACE
MMI SUNNYVALE, CALIFORNIA
CLK A18     A19       A3       A2    A1     /RD /WR     CLKIO  GND
/OC CLKOUT /CLKSTR1 /CLKSTR2 READY /ERROR /GO /GOMSTR CLKPAL VCC

/READY   :=/RD*/WR                       ;READY SIGNAL FED BACK TO 8086
          +/A18                          ;CHIP SELECT MUST BE ENABLED FOR THE
          +/A19                          ;READY SIGNAL TO OCCUR

CLKSTR2 := CLKSTR1* GO                   ;SLAVE CLOCK STRETCH FOR READ CYCLE
         + CLKSTR2* CLKSTR1              ;HOLD CLKSTR2
         + CLKSTR2* CLKIO                ;HOLD CLKSTR2

GO      := GOMSTR* RD* A3* A2* A1        ·GO SIGNAL TO DRIVE S516 DURING READ
         + GOMSTR* WR*/A3                ;GO SIGNAL TO DRIVE S516 DURING WRITE
         + GOMSTR* WR*/A2
         + GOMSTR* WR*/A1
         + GO* GOMSTR                    ;HOLD GO
         + GO* CLKIO                     ;HOLD GO

ERROR   := A19* A18* RD*/A3              ;READ NOT ALLOWED FOR INSTR 0-3
         + A19* A18* RD*/A2              ;READ NOT ALLOWED FOR INSTR 0,1,4,5
         + A19* A18* RD*/A1              ;READ NOT ALLOWED FOR INSTR 0,2,4,6
         + A19* A18* WR* A3* A2* A1      ;WRITE NOT ALLOWED FOR INSTR 7

IF(VCC) /CLKOUT  =/CLKSTR2* CLKIO        ;OUTPUT CLOCK DRIVES S516

IF(VCC) /CLKPAL  = CLKIO                 ;.FUNNY. CLOCK FOR PAL

IF(VCC)  CLKSTR1 = GO* CLKIO* RD         ;SET CLOCK STRETCH MASTER SIGNAL
                 + GO* CLKIO* WR         ;SET CLOCK STRETCH MASTER SIGNAL
                 + CLKSTR1* GO           ;HOLD CLOCK STRETCH MASTER SIGNAL
                 + CLKSTR1*/CLKIO        ;HOLD CLOCK STRETCH MASTER SIGNAL

IF(VCC)  GOMSTR  = RD* CLKIO* A18* A19   ;SET GO MASTER SIGNAL
                 + WR* CLKIO* A18* A19   ;SET GO MASTER SIGNAL
                 + RD* GOMSTR            ;HOLD GO MASTER
                 + GOMSTR*/CLKIO         ;HOLD GO MASTER
                 + GOMSTR* WR            ;HOLD GO MASTER
```

Function Table

```
/OC CLK A18 A19 A3 A2 A1 CLKIO /WR /RD /CLKSTR1 /CLKSTR2 /GOMSTR /GO
CLKPAL CLKOUT READY /ERROR
;                     / /
;                     C C /
;                     L L G   C C   /
;              C      K K O   L L R E
;              L      S S M   K K E R
;/ C A A       K / /  T T S / P O A R
;O L 1 1 A A A I W R  R R T G A U D O
;C K 8 9 3 2 1 O R D  1 2 R O L T Y R    COMMENTS
------------------------------------------------------------------------
 L C H H X X X L H H  H H H H H H L H    INITIALLY NOTHING DONE
 L L H H X X X H H H  H H H H L L L H    READY FOR INSTRUCTIONS
 L C H H L X X L L H  H H H H H H H H    WRITE ASSERTED
 L L H H L X X H L H  H H L H L L H H    GO MASTER ASSERTED
 L C H H L X X L L H  H H L L H H H H    GO SLAVE ASSERTED
 L L H H L X X H L H  L H L L L L H H    CLOCK STRETCH1 ASSERTED
 L C H H X X X L H H  L L L L H H L H    CLOCK STRETCH2 ASSERTED, WRITE NOT
 L L H H X X X H H H  L L H L L H L H    STRETCH CLOCKOUT, GO MASTER REMOVED
 L C H H X X X L H H  L L H H H H L H    STRETCH CLOCKOUT, GO SLAVE REMOVED
 L L H H X X X H H H  H L H H L H L H    STRETCH CLOCKOUT, CLKSTR1 REMOVED
 L C H H X X X L H H  H H H H H H L H    CLOCKOUT NORMAL, TRANSFER COMPLETE
 L L H H X X X H H H  H H H H L L L H    READY FOR INSTRUCTIONS
 L C H H H H H L H L  H H H H H H H H    READ ASSERTED
 L L H H H H H H H L  H H L H L L H H    GO MASTER ASSERTED
 L C H H H H H L H L  H H L L H H H H    GO SLAVE ASSERTED
 L L H H H H H H H L  L H L L L L H H    CLOCK STRETCH1 ASSERTED
 L C H H X X X L H H  L L L L H H L H    CLOCK STRETCH2 ASSERTED, READ NOT
 L L H H X X X H H H  L L H L L H L H    STRETCH CLOCKOUT, GO MASTER REMOVED
 L C H H X X X L H H  L L H H H H L H    STRETCH CLOCKOUT, GO SLAVE REMOVED
 L L H H X X X H H H  H L H H L H L H    STRETCH CLOCKOUT, CLKSTR1 REMOVED
 L C H H X X X L H H  H H H H H H L H    STRETCH CLOCKOUT, CLKSTR2 REMOVED
 L L H H X X X H H H  H H H H L L L H    CLOCKOUT NORMAL, TRANSFER COMPLETE
 H C L H X X X L H H  H Z H Z H H Z Z    HI-Z TEST FOR REG OUTPUTS
 L L H H X L X H H L  H H L H L L L H    ILLEGAL READ
 L C H H X L X L H L  H H L H H H H L    ERROR ASSERTED
 L L H H X X X H H H  H H H H L L H L    ERROR REMAINED
 L C H H X X X L H H  H H H H H H L H    ERROR REMOVED
 L L H H H H H H L H  H H L H L L L H    ILLEGAL WRITE
 L C H H H H H L L H  H H L H H H H L    ERROR ASSERTED
 L L H H X X X H H H  H H H H L L H L    ERROR REMAINED
 L C H H X X X L H H  H H H H H H L H    ERROR REMOVED
------------------------------------------------------------------------
```

Description

This is an interface between an Intel 8086 CPU and a 54/74S516 multiplier. Its primary function is to provide properly timed signals from the 8086 to the 'S516 to increase the efficiency of the 8086. Several design techniques must be employed to accomplish the intricate timing requirements of the interface.

The two most significant address lines from the 8086 must be asserted to enable the PAL, ie. (A19 = A18 = HI). The system clock, running at 5 MHz, is used as an input (CLKIO). From this clock, two other clock signals are generated. CLKOUT is an inversion of the system clock and is used to drive the 'S516. CLKPAL is also an inversion of CLKIO but it is fed back to pin 1 of the PAL and used to drive the registered outputs of the PAL. Read ($\overline{\text{RD}}$) and write ($\overline{\text{WR}}$) are signals from the 8086 which are used to drive the 190 input of the 'S516.

To generate the Go signal for the 'S516, the PAL uses the master/slave principal to provide proper timing. Go Master is asserted ($\overline{\text{GOMSTR}}$ = LOW) which in turn asserts Go Slave ($\overline{\text{GO}}$ = LOW). This signal ($\overline{\text{GO}}$) is then sent to he 'S516 to perform the desidered operation.

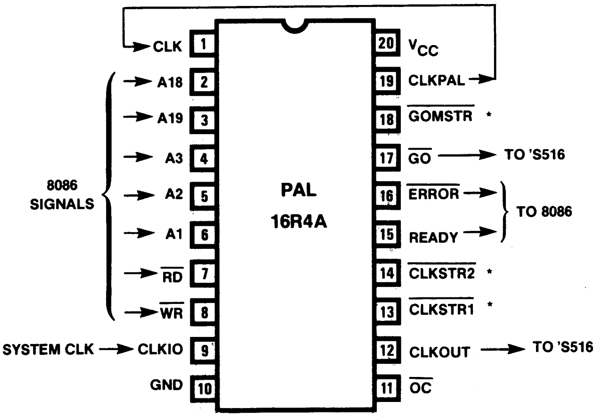
During a Read ($\overline{\text{RD}}$) operation, data is clocked out of the 'S516 in one cycle; however, it takes two cycles for the 8086 to latch in the data. To accomplish this, the 'S516 clock is stretched high (CLKOUT = HI) for two cycles. Again, the master/slave principal is used to provide a properly timed clock stretch. Clock Stretch Master is asserted ($\overline{\text{CLKSTR1}}$ = LOW) which in turn asserts Clock Stretch Slave ($\overline{\text{CLKSTR2}}$ = LOW) which holds CLKOUT = HI for the desired interval.

Ready (READY = HI) is sent from the PAL back to the 8086 to acknowledge Read or Write.

An error flag will be asserted for illegal write (write with instruction 7) or illegal read (read with instruction 0-6).

```
8086 CPU/74S516 MULTIPLIER INTERFACE

                11 1111 1111 2222 2222 2233
  0123 4567 8901 2345 6789 0123 4567 8901

 0 ---- ---- ---- ---- ---- ---- ---- ----
 1 ---- ---- ---- ---- ---- ---- ---- X--- CLKIO

 8 ---- ---- ---- ---- ---- ---- ---- ----
 9 X--- X--- ---- ---- ---- -X-- ---- X--- RD*CLKIO*A18*A19
10 X--- X--- ---- ---- ---- ---- -X-- X--- WR*CLKIO*A18*A19
11 ---- ---X ---- ---- ---- -X-- ---- ---- RD*GOMSTR
12 ---- ---X ---- ---- ---- ---- ---- -X-- GOMSTR*/CLKIO
13 ---- ---X ---- ---- ---- ---- -X-- ---- GOMSTR*WR

16 ---- ---X X--- X--- X--- -X-- ---- ---- GOMSTR*RD*A3*A2*A1
17 ---- ---X -X-- ---- ---- ---- -X-- ---- GOMSTR*WR*/A3
18 ---- ---X ---- -X-- ---- ---- -X-- ---- GOMSTR*WR*/A2
19 ---- ---X ---- ---- -X-- ---- -X-- ---- GOMSTR*WR*/A1
20 ---- ---X ---X ---- ---- ---- ---- ---- GO*GOMSTR
21 ---- ---- ---X ---- ---- ---- ---- X--- GO*CLKIO

24 X--- X--- -X-- ---- ---- -X-- ---- ---- A19*A18*RD*/A3
25 X--- X--- ---- -X-- ---- -X-- ---- ---- A19*A18*RD*/A2
26 X--- X--- ---- ---- -X-- -X-- ---- ---- A19*A18*RD*/A1
27 X--- X--- X--- X--- X--- ---- -X-- ---- A19*A18*WR*A3*A2*A1

32 ---- ---- ---- ---- ---- X--- X--- ---- /RD*/WR
33 -X-- ---- ---- ---- ---- ---- ---- ---- /A18
34 ---- -X-- ---- ---- ---- ---- ---- ---- /A19

40 ---- ---- ---X ---- ---- ---- ---X ---- CLKSTR1*GO
41 ---- ---- ---- ---- ---- ---X ---X ---- CLKSTR2*CLKSTR1
42 ---- ---- ---- ---- ---- ---X ---- X--- CLKSTR2*CLKIO

48 ---- ---- ---- ---- ---- ---- ---- ----
49 ---- ---- ---X ---- ---- -X-- ---- X--- GO*CLKIO*RD
50 ---- ---- ---X ---- ---- ---- -X-- X--- GO*CLKIO*WR
51 ---- ---- ---X ---- ---- ---- ---X ---- CLKSTR1*GO
52 ---- ---- ---- ---- ---- ---- ---X -X-- CLKSTR1*/CLKIO

56 ---- ---- ---- ---- ---- ---- ---- ----
57 ---- ---- ---- ---- ---- --X- ---- X--- /CLKSTR2*CLKIO

LEGEND:  X : FUSE NOT BLOWN (L,N,0)    - : FUSE BLOWN   (H,P,1)

NUMBER OF FUSES BLOWN =  919
```

The PAL Used in the 8086 CPU/74S516 Multiplier Interface



* Used Internally in PAL

Doing Your Own Thing in High-Speed Digital Arithmetic*

Chuck Hastings

Some problems which seem, right off, to be too big for even the latest microprocessor-based small computing systems aren't, really — if they're approached right, they can be cut down to size. The basic idea is to divide the problem up into a complex part which isn't executed at high speed, and a simple and repetitive part which is.

Microprocessors are good at handling complexity. If the high-speed, repetitive tasks can be unloaded onto some more specialized logic under the control of the microprocessor, the combination may be able to keep up with real-time jobs which would seem to call for a large minicomputer. In the particular case where the multiply and/or the divide instructions of the microprocessor are too slow to allow the job to be done in real time, or where it doesn't even have these instructions, this "specialized logic" may sometimes consist of just one chip!

This tutorial paper presents in detail two of the standard tricks of the trade in high-speed arithmetic: carry prediction and bypassing, and Booth multiplication. Hopefully, these have been presented in such a way that you don't have to be a veteran logic designer to fully comprehend both the presentation and the potential value of the techniques to you. The emphasis will be on gaining *understanding* of these techniques, but there will also be some information on actual products which incorporate them --the 54/74S182 carry bypass, the 54/74S381/2 ALUs, and the 54/74S508 and 54/74S516 multiplier/accumulator/dividers.

This paper is as nearly self-contained as I could make it. Although it grapples with some moderately heavy topics, it develops whatever notions and notation are needed to address these topics as it goes along. You can probably follow the presentation if, say, you already feel that you clearly understand what a binary number is. And if you actually know quite a lot more than that, I hope you will still find that the rather unconventional derivations presented here provide some useful insights.

* This paper is a slightly modified version of the paper by the same name which appeared in the *Conference Proceedings of the 6th West Coast Computer Faire,* pages 492-510; 3-5 April 1981.

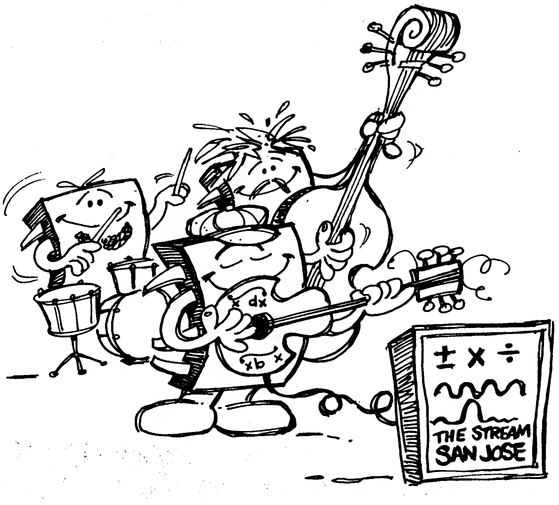
Doing Your Own Thing in High-Speed Digital Arithmetic

Chuck Hastings

Division of Labor in Systems

To paraphrase Parkinson's Law, work expands to fill the computing capacity available. There are always applications, which are just a little bit beyond the research of available (or affordable!) hardware, which could be handled by your computer system or mine If Only . . . A traditional system-designer strategy for pulling a few of these If Onlys into the realm of the technically and financially feasible is to divide up the task to be done into a general-purpose part and a special-purpose part. The general-purpose part then gets implemented on a general-purpose computer, or today very likely on a microcomputer, whereas the special-purpose part gets implemented by means of some hardware tailored explicitly to do it quickly and cost-effectively. If, as is often the case, the job can be broken up into complex but fairly low-repetition-rate tasks and simple but high-repetition-rate tasks, this strategy works well.

In years gone by, systems configured according to this strategy were sometimes referred to as "hybrid" or "fixed-plus-variable-structure" — the latter particularly when the custom hardware was subject to rework when the system changed from one application to another. Today, sometimes the custom hardware is itself a processor capable of being programmed or microprogrammed, but specialized to be good at something which a general-purpose processor isn't good at. The obvious typical example is the "array processor" type of machine, which is sold as an attachment or peripheral for a minicomputer and isn't intended to function as a free-standing processor itself. In this case, the team of minicomputer plus array processor may have computing capabilities literally up in the Cray-1 range when used shrewdly to solve problems to which it is well adapted, at a fraction of the cost of a Cray-1 — even though the minicomputer by itself would be orders of magnitude slower on the same problem. (See Reference (1).)



Existing array processors are primarily arithmetically oriented. The astonishing recent financial success of several array-processor vendors would seem to indicate that there really are some problems out there whose "high-repetition-rate part" is basically arithmetic. If you have some personal or professional interest in solving such problems, by now you should be motivated to read the rest of this paper despite the appearance, now and then, of an occasional mathematical equation.

One final bit of systems-level philosophizing before we plunge into that, however. It will be necessary to *interface* your general-purpose small computer to your special-purpose logic. There is an occasional situation (for instance, when using the 54/74S516 multiplier/divider chip which I'll describe later) in which much of this has already more or less been done for you. Interfacing techniques are fairly well documented these days in applications literature from microprocessor manufacturers, and are a large digression from my major topic of fast digital arithmetic, so I'll say only a few words about them. Conceptually there are two major ways to treat a hunk of special-purpose logic for interfacing purposes: as a peripheral, or as part of "memory" within the processor's addressing space. The second approach is generally more in vogue today; of the many systems which do it that way, I'll mention the Digital Equipment Corporation PDP-11 family of computers and the Motorola 6800 line of microprocessors and support chips. For a general, not-too-detailed discussion of several different ways of interfacing the 'S516 to microprocessors and bit-slice processors, see reference (2). For some additional information on 'S516 interfacing methods, plus a brief analysis of one economically-important mini-number-cruncher application, see reference (3).

Carry Prediction and Bypassing

A binary full adder is a circuit, or a portion of a circuit, capable of adding together two one-bit numbers A and B to form a two-bit sum. Since one-bit numbers by themselves aren't a whole lot of use, a one-bit full adder is generally a stage of a much longer adder, and only the least-significant bit of this two-bit sum is referred to as the "sum bit" S. Since the most-significant bit "carries over" into the next stage to the left, it is of course the "carry bit" C. (You learned all about "carries" in elementary school, didn't you?) Obviously, subscripts are needed to tell the bits associated with a given stage of a multi-stage adder apart from those associated with some different stage. I will use subscripts in equations, but I won't use them much in the main text.

Naturally, the carry out of stage i is the carry into stage i+1, and must be added together with the A and B bits at that stage to yield both the S bit for stage i+1 and the carry out of stage i+1 and into stage i+2. Hence, all full-adder stages above the least-significant one must actually be able to add together *three* bits — the A and B bits for that stage, and the C bit from the preceding stage. In practice, there even needs to be a carry into the least-significant stage also. Thus, a real-world "full adder" *always* must be able to add together three bits and not just two bits.

The reason why there must be this carry into the least-significant adder stage is that it gets manipulated by the control logic in order to make the same adder also be able to subtract. Subtracting B from A is normally done by *complementing* all the bits of B (which means reversing 0's to 1's and 1's to 0's), and forcing that least-significant "carry-in" to be a 1. But I digress.

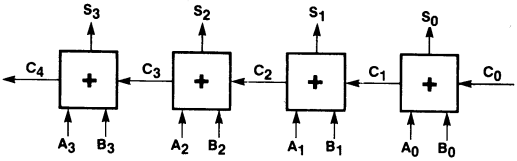


Figure 1. Ripple-Carry Adder

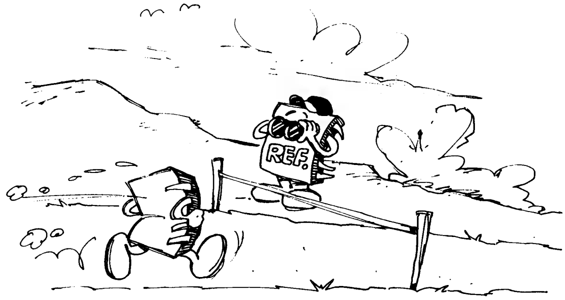
Figure 1 is a diagram of the multi-stage adder which we have just verbally walked through. By now it is all too apparent what is the "critical path" in this logic — the most time-consuming path over which signals must travel before a correct answer is available and electrically stable at the adder's sum outputs. At *each* stage, the carry out of that stage depends on the carry into that stage. Since the carry signals from one stage to the next can be considered as a wave of carry information "rippling" all the way from the least-significant end of the adder to the most-significant end, the whole thing is often referred to as a "ripple-carry adder." Practical adders may have 32, 48, 60, 64, or even 72 stages; and so, obviously, this rippling process can take a while, as time is measured by impatient digital-system designers. For instance, if a 64-bit ripple-carry adder were to be constructed using low-complexity 74H183 full-adder chips, which have a worst-case logic-delay time of 18 nanoseconds for propagation of a carry from input to output, conservative design practice would imply that at least 64x18 = 1,152 nanoseconds, or more than a microsecond, would have to be allowed for the adder outputs to stabilize with the correct result available — and designers today expect to be able to perform a 64-bit addition operation in less than 100 nanoseconds! Which leads to the question, can we look ahead and see what carry signals are coming before they arrive? After all, the entire situation *is* determined in advance — if A and B were small-enough numbers, say at most 7 bits each, their sum could just be looked up in a memory having a 14-bit address, i.e., having 16,384 locations. It is only the *economics* of memory which make that approach grossly infeasible as the word length of A and B increases!

". . . THE WHOLE THING IS OFTEN REFERRED TO AS A "RIPPLE-CARRY ADDER" . . ."

It turns out, of course, that we indeed can look ahead. To see how this is done, we concoct two more bits which can be evaluated at each stage of an adder in terms of A and B, *without* taking C into account. After all, C has to ripple all the way up from the least-significant end! These are:

$$G_i = A_i \& B_i$$

$$P_i = A_i \mathrm{v} B_i$$



". . . CAN WE LOOK AHEAD AND SEE WHAT CARRY SIGNALS ARE COMING BEFORE THEY ARRIVE? . . ."

The G bit is the Generate condition — a "1" for G at stage i means that there will be a carry out of that stage even if there is no carry into that stage, which is to say, that a carry will be *generated* at that stage. The P bit is the Propagate condition — a "1" for P at stage i means that a carry into that stage will elicit a carry out of that stage, which is to say, *propagate* through the stage. The symbols "&" and "v" are used here for the "Boolean" (logical) "and" and "or" operators respectively. (You'll run into the term "Boolean" in computer logic-design papers, as a synonym for "formal logical." The modern approach to formal logic was devised by an Irish cleric and mathematician named George Boole in 1854 in a famous book entitled *Laws of Thought*, which is available from Dover Books.) Thus, G is "1" if and only if *both* A is "1" *and* B is "1," whereas P is "1" if *either* A is "1" *or* B is "1." Various other choices may be made for typographic symbols for these two operators, but this choice has the advantage of leaving the symbol "+" free to denote actual addition. ("+" often gets used in written papers on logic design to denote the Boolean "or" operator.)

We need one more Boolean operator. This one is called "exclusive-or," to distinguish it from the garden-variety plain "or" just introduced; to be *very* proper, plain "or" is sometimes called "inclusive-or." The bit "A exclusive-or B" is defined to be "1" if and only if *either* A is "1" *or* B is "1" — *but not both.* The "but-not-both" stipulation is what distinguishes "exclusive-or" from "inclusive-or." To evaluate "exclusive-or," we must be able to denote the complements of A and B respectively by putting bars over them. Then,

$$A_i \,\$ B_i = (A_i \,\&\overline{B}_i) \mathrm{\ v\ } (\overline{A}_i \,\&B_i)$$

Use of the symbol "$" for "exclusive-or" is ridiculously nonstandard, but my typographic alternatives are rather limited. Anyway, we are now ready to understand Tables 1 and 2:

			$C_i + B_i + A_i$	
C_i	B_i	A_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 1. What a Full Adder is Supposed to Do

Notice that the two result bits on the right form, as promised, a two-bit number which is simply the *sum* of the three bits on the left interpreted as one-bit numbers and added arithmetically — that is, the *count* of all the "1" bits present in the three input bits C, B, and A at stage i.

Next, we present a way of determining the correct values for the stage-i sum, and the "carry-out" of stage i, in terms of the propagate-and-generate formulation:

						C_{i+1}		S_i
C_i	B_i	A_i	G_i	P_i	$P_i \,\&C_i$	$G_i \mathrm{\ v}(P_i \,\&C_i)$	$G_i \,\$P_i$	$(G_i \,\$P_i)\,\$C_i$
0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	1
0	1	0	0	1	0	0	1	1
0	1	1	1	1	0	1	0	0
1	0	0	0	0	0	0	0	1
1	0	1	0	1	1	1	1	0
1	1	0	0	1	1	1	1	0
1	1	1	1	1	1	1	0	1

Table 2. Full-Adder Sum and Carry-Out Expressed in Terms of Propagate and Generate

As you can see, the columns for the sum and carry-out of stage i in Table 2 are identical with those in Table 1 — only now, in Table 2, we *also* have a method for evaluating these outputs in terms of basic Boolean operators, rather than just an intuitive justification for them. This is an important practical step, since each Boolean operator generally turns into one "gate" circuit in a realistic electronic implementation. In an LSI (*L*arge-*S*cale *I*ntegration) circuit such as the 'S516, one gate is apt to require just a few "square mils" of silicon area. (One "mil" is 1/1,000 of an inch, so one "square mil" is 1/1,000,000 of a square inch.)

It should be remarked in passing that:

$$G_i \,\$P_i = A_i \,\$B_i$$

since you may run into that statement if you get inspired to read logic-design textbooks, and want to relate what they say to what I've said here.

By matching up the appropriate columns of Tables 1 and 2, we have verified the correctness of the following equations:

$$S_i = (G_i \,\$P_i)\,\$C_i$$

$$C_{i+1} = G_i \mathrm{\ v}(P_i \,\&C_i)$$

The equation for the sum bit is important since it provides our link to the final desired answer, but for now the equation for the carry-out is the key one. This same notion of generate and propagate, which by now we have pretty well beaten to death for one stage of a full adder, may be extended to encompass the generate and propagate conditions for groups of *several* full-adder stages each, by the mathematical process called "recursion." By doing this, we can derive a logic equation which can "predict" (with 100% accuracy) the carry-out of a group of n full-adder stages given the carry-in and the A and B bits for each of the stages. Hence, a circuit which implements this carry-prediction equation can in effect "bypass" all of those full-adder stages when one computes the overall carry-rippling time for the whole adder.

Recursion in developing carry-prediction equations is something like playing leapfrog. First, we take the above equation for the carry-out and substitute i+1 for i in the subscripts, which gives:

$$C_{i+2} = G_{i+1} \mathrm{\ v\ } (P_{i+1} \,\&C_{i+1})$$

At this point it is expedient to again switch notation, basically so we can crowd more stuff onto one line. So the mere juxtaposition (writing right next to each other) of two Boolean variable symbols such as G, P, etc., will now be understood to imply that the "and" of these symbols is being expressed, and the parentheses around such a Boolean "and" of two or more variables will be omitted. (Don't say I didn't warn you!) Our two carry-out equations now read:

$$C_{i+1} = G_i \mathrm{\ v\ } P_i C_i$$

$$C_{i+2} = G_{i+1} \mathrm{\ v\ } P_{i+1} C_{i+1}$$

Substituting the carry-out from the first equation into the second equation as the carry-in gives:

$$C_{i+2} = G_{i+1} \mathrm{\ v\ } P_{i+1} G_i \mathrm{\ v\ } P_{i+1} P_i \, C_i$$

What we now have is a carry-prediction equation over two adder stages. Comparing this equation with that for the one-stage case, we see that the expression:

$$G_{i+1} \vee P_{i+1} G_i$$

may be interpreted as a two-stage "group generate," and that the term:

$$P_{i+1} P_i$$

may likewise be interpreted as a two-stage "group propagate." If we continue on with this process up to the economically-important case of four full-adder stages, and simplify our notation further by pegging "i" at stage zero so that i becomes 0, i+1 becomes 1, etc., we get the whole raft of equations in Table 3. (I'll spare you the gory details, but you may of course wish to work them out for yourself just to prove I'm not lying to you!)

$$\mathbf{C_1} = G_0 \vee P_0\, C_0$$

$$\mathbf{C_2} = G_1 \vee P_1\, G_0 \vee P_1\, P_0\, C_0$$

$$\mathbf{C_3} = G_2 \vee P_2\, G_1 \vee P_2\, P_1\, G_0 \vee P_2\, P_1\, P_0\, C_0$$

$$\mathbf{C_4} = G_3 \vee P_3\, G_2 \vee P_3\, P_2\, G_1 \vee P_3\, P_2\, P_1\, P_0 \vee P_3\, P_2\, P_1\, P_0\, C_0$$

$$\mathbf{G} = G_3 \vee P_3\, G_2 \vee P_3\, P_2\, G_1 \vee P_3\, P_2\, P_1\, P_0$$

$$\mathbf{P} = P_3\, P_2\, P_1\, P_0$$

Table 3. Carry-Prediction Equations for a Four-stage Full-Adder Carry Bypass Circuit

The equations for the four-stage group-generate condition G and the four-stage group-propagate condition P were, of course, obtained by inspection by factoring the equation for C_4. So, obviously,

$$C_4 = G \vee PC_0$$

I need to make some out-of-context remarks here, so that you don't get confused later. The way my subscripts are operating is that the least-significant bit position has subscript "0," the next one to the left has subscript "1," etc. The subscripts in Table 3 are for the adder stages which are used with a first-level bypass, which in real-life commercial circuits means the bypass which is buried within a chip which also contains four full adders (see below). For second-level bypasses, the subscripts go up in jumps of four. For a third-level bypass, they jump up by 16. Where multiple adders and/or bypass circuits are used in one system, it is apt to be necessary to mentally add 16, 32, 48, etc. to the subscripts used for the carry outputs of parts toward the most-significant end. My use of subscripts conforms to normal practice in the computer-design world, but quite different things are done in diagrams on data sheets from semiconductor manufacturers.

". . . RECURSION IN DEVELOPING CARRY-PREDICTION EQUATIONS IN SOMETHING LIKE PLAYING LEAPFROG . . ."

Practical Circuits

We have reached a point where the operation of standard commercial integrated circuits which incorporate carry bypassing should begin to be comprehensible. Table 4 summarizes these.

The '182-class circuits and the 10179 are considered as building-block components, with the G and P outputs made available so that "third-level bypassing" may be used. (More on this topic in just a minute.) The rest are considered as components which provide *all* necessary carry bypassing for a mill of length at most 32 bits. The reason why it is ok to only supply every fourth carry, or even every eighth carry, needed by the adders is that commercial adders come in blocks of four, and only the carry into the least-significant adder of these four needs to be supplied externally. (You'll see how all that works in just a minute, too.) My notation in Table 4 is fitting and proper within the context of this paper, and some of the newer data sheets use it, but I don't guarantee that it is the same notation you will encounter when you start reading data sheets.

In the three cases (74AS882, 10179, and 100179) where only every eighth carry rather than every fourth carry gets provided as an output, what happened is that for some reason there weren't enough pins left on the circuit package used by the semiconductor manufacturer to provide the other carries. When using one of these circuits, the designer either has to use two of them in place of one in every position and "stagger" these by one input G-and-P group with respect to each other, or else use the carry outputs available on some full-adder circuits (usually called ALUs, or *A*rithmetic *L*ogic *U*nits) to fill in the missing in-between carries. At the least-significant end of any mill where two staggered carry-bypass circuits are used, one stage of one of these bypass circuits must be connected up so that it always passes a carry but never generates one — that is, so that it in effect isn't there. The required hard-wired input connections are:

$$G_0 = 0 \text{ and } P_0 = 1$$

TYPE NUMBER(S)	CIRCUIT TECHNOLOGY	OUTPUTS PROVIDED BY ONE CHIP
74182, 74LS182, 74S182, 2902	TTL	G, P, C_{12}, C_8, C_4
74AS882	TTL	$C_{32}, C_{24}, C_{16}, C_8$
3003	TTL	$C_{32}, C_{28}, C_{24}, C_{20}, C_{16}, C_{12}, C_8, C_4$
10179	ECL	G, P, C_{16}, C_8
100179	ECL	$C_{32}, C_{24}, C_{16}, C_8$

Table 4. Commercial Carry-Bypass Integrated Circuits

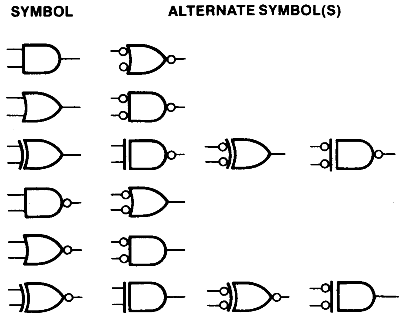
To understand the schematic for a carry-bypass circuit, we first have to introduce some symbols, which if properly used provide a way of diagramming logic equations which is every bit as exact as the equations themselves. Logic symbol conventions are a subject about which many learned people learnedly disagree, and about which there is generally several times as much confusion as there needs to be. The scheme I will use is a slight extension of the quasi-standard "MIL-806B" conventions devised by the military almost two decades ago, to represent "mixed logic." The idea here is that the most "natural" situation is that a higher ("H") voltage level signifies that the Boolean variable at that circuit point is to be understood as a "1," and that a lower ("L") level correspondingly signifies that the variable is to be understood as a "0." This situation is called "positive logic" or, more precisely, "assertive-high" logic. The opposite situation is assumed to be less common but is entirely permissible, where "H" goes with "0" and "L" goes with "1"; this situation is, not surprisingly, called "negative logic" or "assertive-low logic." (Also, the word "active" is sometimes used instead of the word "assertive.") "Mixed logic" then means a set of diagramming conventions in which assertive-high logic and assertive-low logic can coexist. At points on the diagram where signals are understood as assertive-low, a small circle (known as a "bubble" in computer-industry jargon) is placed at the point where a line (path) encounters a gate symbol. When a signal proceeds along a path and does not change electrically, but the designer's perception of its assertiveness does change, a "psychological inversion" symbol is used on the path to mark the point where the perception changes. Reference (4) recommends a small line (a "stroke") across the path, but this "stroke" does not show up well on blue-line prints, and so I use a small wedge (a "dagger") pointing at the line and right up against it, and blacked in; this wedge is a symbol usually available on logic templates.

There are really three basic two-input gate circuts: "and," "or," and "exclusive-or" or "ex-or." With the modification of a bubble at the output, these three turn into three more which are also pretty basic: "nand," "nor," and "ex-nor" or "coincidence." There are two ways (four for ex-or and ex-nor) of representing each of these basic gate circuits with MIL-806B-like symbols. You may verify that each way produces the same basic Boolean-electrical behavior, represented by the small table over at the right:

In passing, you may notice that the name "coincidence" for the ex-nor gate comes from the fact that its output is high whenever its two inputs "coincide," that is, are equal.

Gates are usually drawn apart, connected by lines. In the special cases of "and-or" and "or-and" compound gates, they may be drawn as "clusters."

Some people, including those who prepare data books for semiconductor companies, don't draw these compound gates as "clusters," but as separate "and" and "or" symbols connected by short lines. However, the cluster symbolism is both easier to grasp and more indicative of the way actual gate structures within integrated circuits (at least, the TTL and ECL integrated circuits with which I am familiar) are designed to work.



GATE	BEHAVIOR FOR BA = HH	HL	LH	LL
AND	H	L	L	L
OR	H	H	H	L
EX-OR	L	H	H	L
NAND	L	H	H	H
NOR	L	L	L	H
EX-NOR	H	L	L	H

Figure 2. Basic Mixed-Logic Gate-Circuit Symbols

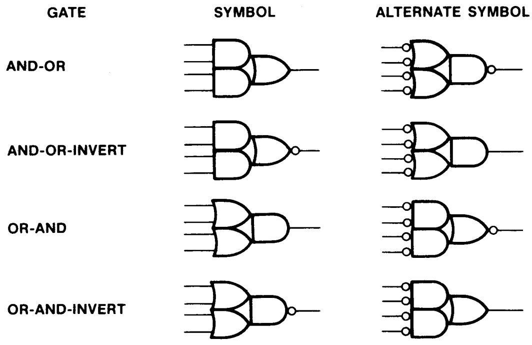


Figure 3. Common Compound Gates

After all that, the schematic diagram for the widely-used 74S182 TTL carry-bypass circuit is almost an anticlimax:

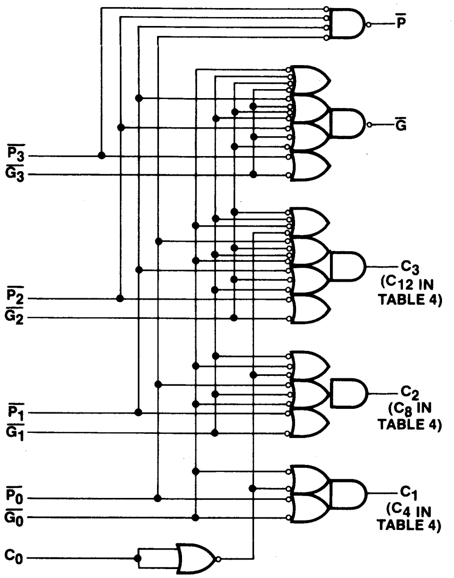


Figure 4. 74S182 Carry-Bypass Circuit (Assertive-Low Representation)

In case you decide to look at a data sheet for this part, I'll also draw it the way it is usually drawn:

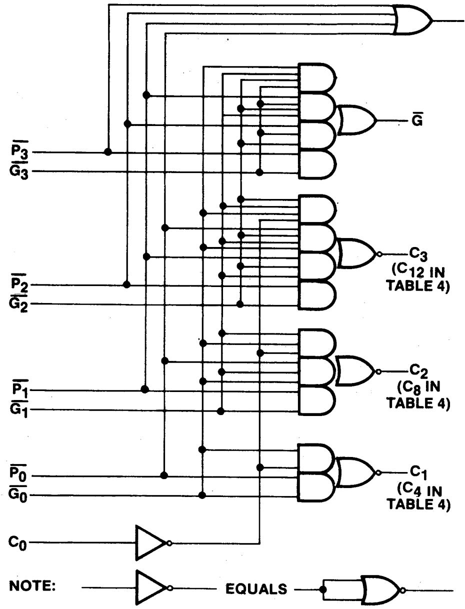


Figure 5. 74S182 Carry-Bypass Circuit (Usual Data-Sheet Representation)

You will also have noticed by now that this circuit expects the generate and propagate inputs it uses to be in assertive-low form, and produces higher-level generate and propagate outputs in assertive-low form also, but produces carry outputs in assertive-high form. This method allows the use of optimal forms of TTL circuit logic on-chip, based on and-or-invert gate structures.

Also, I have to admit to one sneaky logic-equation manipulation trick in getting from the equations of Table 3 to the logic of Figure 4 (where it is easy to follow) and then to Figure 5 (where it isn't). I'll illustrate this trick for the simplest case, which is the logic for C_1 and corresponds to the bottom gate cluster in Figure 4:

$$C_1 = G_0 \vee P_0C_0 = (G_0 \vee P_0)(G_0 \vee C_0)$$

where considerable use has been made of the "Law of Absorption" from Boolean algebra, which states that

$$A \vee AB = A$$

and of the identity

$$AA = A$$

The refactorization of the logic for C_2, C_3, and C_4 so that it looks like the parenthesis form on the right above is similar to that for C_1, only messier.

When you begin looking at data sheets, you will also notice that most of the ALUs (and also the more complex "bit-slice" registered ALUs such as the 2901) incorporate gate structures very much like those of the 74S182 *within* the chip, for generating both the internal carries needed therein and the generate and propagate signals which are output for use by an external carry-bypass chip.

There are too many different ALU types for me to dig very far into this topic here. Probably the most commonly used ones today are the 74S181 and 74S381 (very fast TTL), the 74LS181 and 74LS381 (lower-power TTL), and the 10181 (ECL). "Bit slices" such as the 2901 (TTL) and 10800 (ECL), which include considerably more functionality and take up correspondingly more circuit-board area, are also often used in current designs.

Designing with Carry-Bypass Circuits

All of the ALUs just enumerated have in common that they perform either arithmetic or logic operations, according to an opcode presented to them at special opcode inputs, on two different four-bit operands, and that they can be catenated ("cascaded") to form a super-ALU of any length which is a multiple of four bits. For clarity, I'll use the British term "mill" (which is short and sweet and unique, and goes all the way back to Babbage) for such a super-ALU. (Don't confuse "mill" with "mil," which — remember? — is 1/1,000 of an inch. Isn't English a wonderful language?) Also, all of these various ALUs also provide group-generate and group-propagate outputs. As you have begun to suspect by now, this means that an external carry-bypass circuit can be connected up with four of them in such a way as to provide optimum-speed carry paths.

To avoid circumlocutions, I'll describe the interconnection strategies for mills of several lengths as they apply to the TTL 74S381 ALU and the 74S182 carry-bypass circuit. These two chips are fairly simple and straightforward, so that we won't get tangled up in other issues besides carry bypassing. (Besides, my employer just happens to make both of them!) Pinouts for both of these chips are in Figure 6; you will notice minor difference in the notation used for the carry outputs, but they are the same outputs you are familiar with by now. Figure 7 shows the simplified block symbols which I will use in interconnection diagrams. For reasons of diagramming simplicity, a 74S182 is always drawn bigger on diagrams (and a "third-level" 74S182 gets drawn bigger still!) than is a 74S381, even though it is actually a littler chip in a littler (16 pins as compared to 20) package!

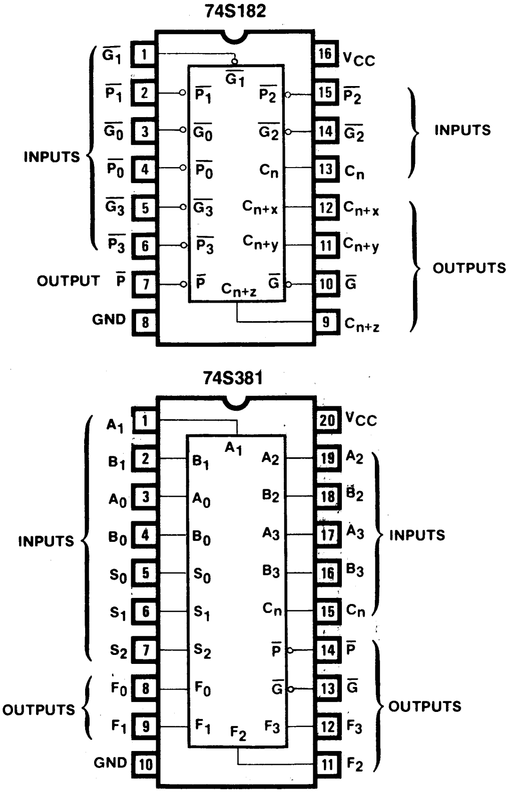


Figure 6. 74S182 and 74S381 Pin Configurations

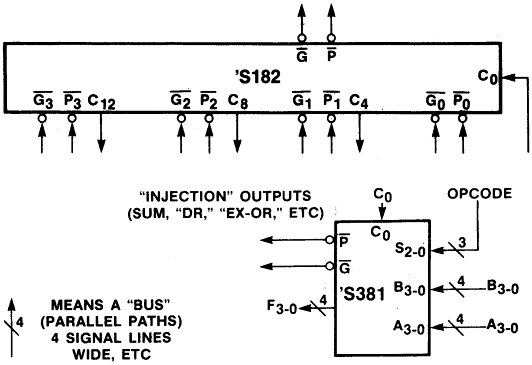


Figure 7. 74S182 and 74S381 Logic Symbols

Although it would take too long for me to walk you through all of the design and operational features of the 74S381 the way we did for the 74S182, here is the opcode table, which will give you some idea of what it can do:

OPCODE BITS			ARITHMETIC OR BOOLEAN OPERATION
S_2	S_1	S_0	
L	L	L	Clear (make all F outputs Ls)
L	L	H	Inverse Subtract (B-A)
L	H	L	Subtract (A-B)
L	H	H	Add (A+B)
H	L	L	Exclusive-Or (A$B)
H	L	H	Or (AvB)
H	H	L	And (A&B)
H	H	H	Preset (make all F outputs Hs)

Table 5. 74S381 Operation Codes and Functions

These operations cover most of the more usual tasks for an ALU. The 74S181, an older design, has five opcode pins instead of three and by my reckoning has 21 usable operations instead of 8 (it also has 11 "garbage" operations!); but for that you must use a package which is physically at least twice as big and give up the Inverse Subtract operation, which it does not have.

In case you were wondering what the carry-prediction network within an ALU does when the ALU is supposed to be performing a Boolean operation and carries would cause an incorrect answer, rest assured that when the ALU is receiving one of the opcodes for a Boolean operation the carry-prediction-network outputs either are forced to zero or else are prevented in some other way from affecting the result.

Also worth mentioning here is the 74S382 modification (a "metal option" in Silicon-Valley jargon) of the 74S381, in which the group-generate and group-propagate outputs are respectively replaced by an output indicating "overflow" condition and by C_4 — which the 74S381 doesn't have. The opcodes don't change. At least two companies, Fairchild and Monolithic Memories, have speeded-up versions of both of these ALUs in design.

Without further ado, Figures 8, 9, and 10 show the interconnection schemes for mills of length 16, 24, and 64 bits, revealing most of the different things which can happen, including "third-level" bypassing. These are drawn in normal arithmetic right-to-left order, the way you learned to add numbers and propagate carries in elementary school, rather than in the quaint left-to-right order often fancied by semiconductor manufacturers in data books. You'll notice that the most-significant ALU is a 74S382 in each case, so that the left-end carry from the entire mill is available, along with the overflow indication, which is formed *within* the 74S382 as:

$$\mathrm{OVR} = C_4\$C_3$$

Negative-number representation and a derivation of this equation are outside the scope of this paper, but the OVR signal is basically there to tell you when you have created a garbage number because the real result of your operation won't fit into the word length of your mill.

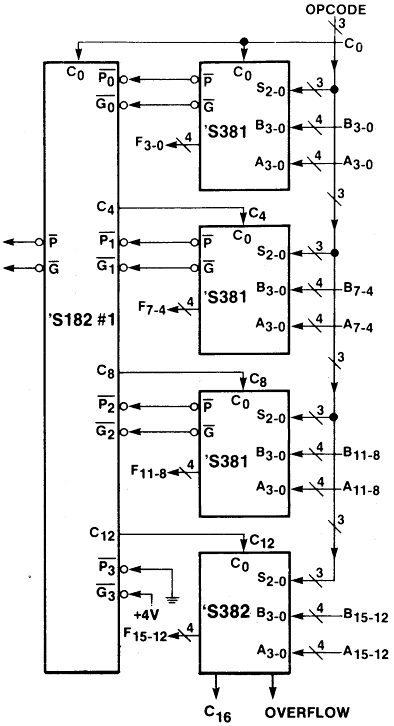
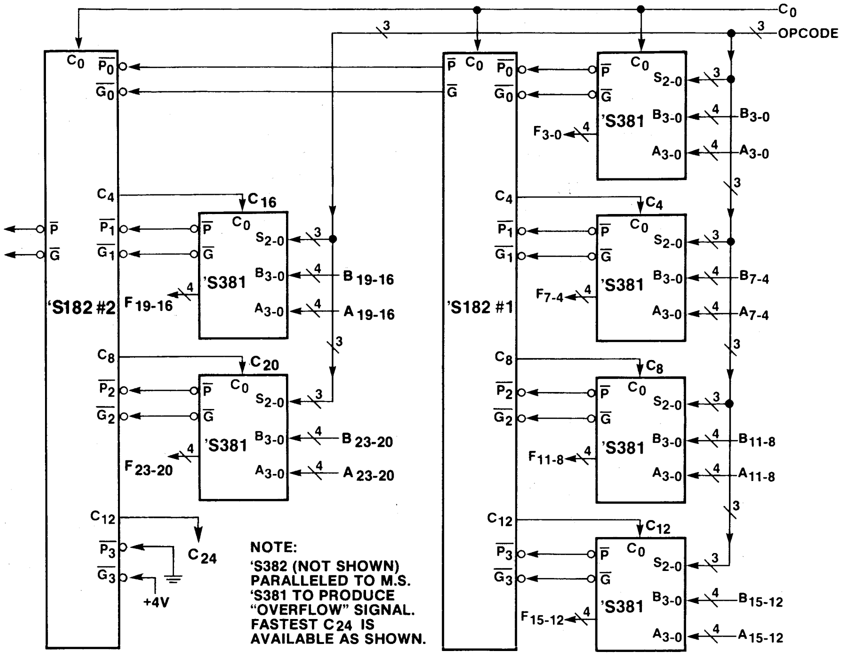


Figure 8. 16-Bit Mill

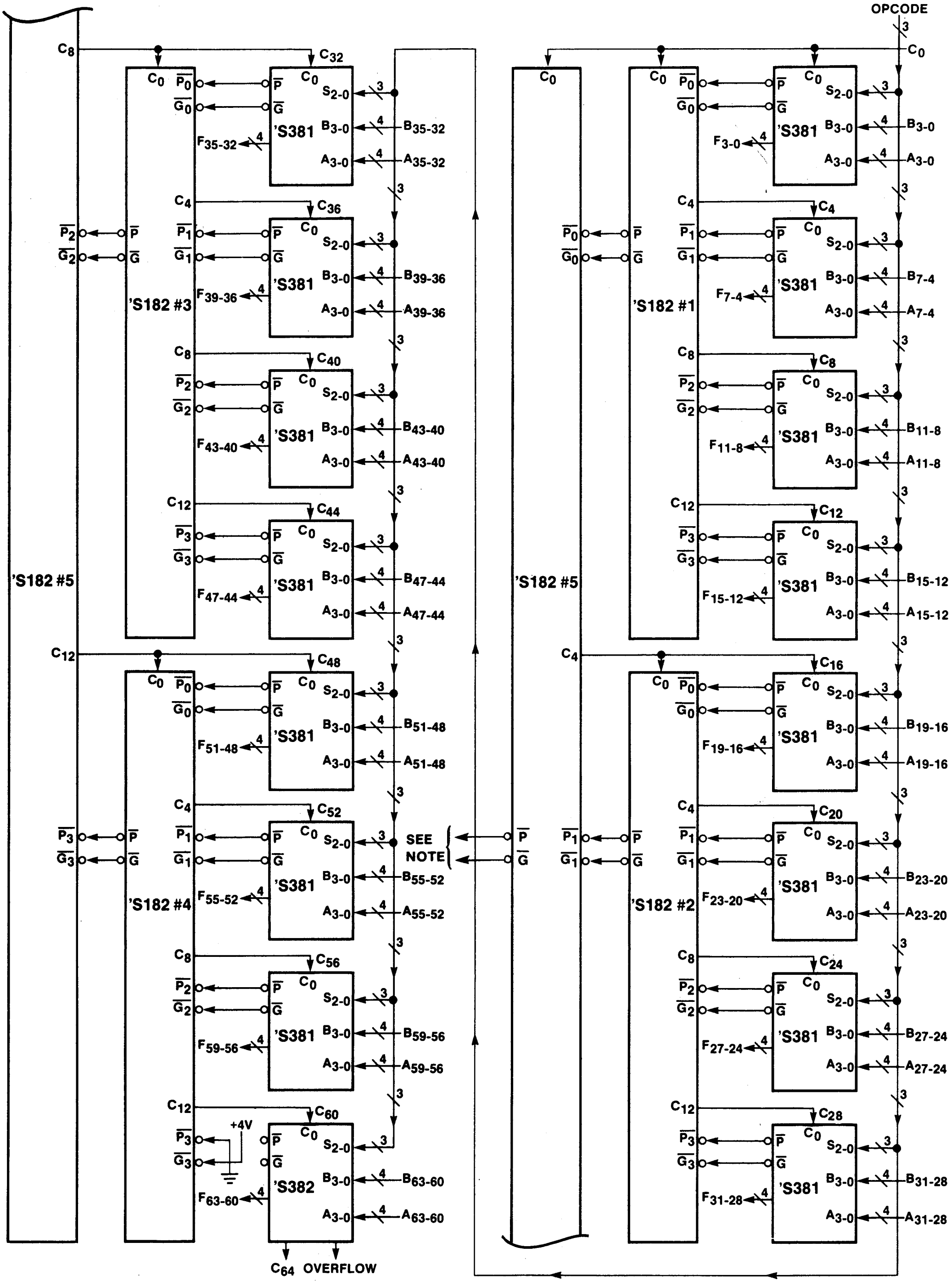
A few final comments on these figures. When the group-generate and group-propagate outputs of a 74S182 aren't going to be needed, it isn't necessary to drive its G_3 and P_3 inputs, since those inputs don't enter into the equations for any of the other outputs. (If you don't believe me, go back and check; the 74S182 *has* no C_{16} output!) Or you can tie input G_3 to a logic zero (implying that no carry is to be generated) and P_3 to a logic one (implying that a carry is always to be propagated), as was already discussed for G_0 and P_0, remembering that these signals are assertive-*low* so that "0" is "H" and "1" is "L." (I can't easily put complementation bars over assertive-low signals in the text, but I do it in schematics, and they should really be there.) This latter method is greatly to be preferred; it really isn't very cool design practice to leave TTL inputs just dangling and unconnected to anything. Also, the approach you may use to determine the effective speed of a mill is suggested in Figure 11, which is for the same mill as Figure 8 except for the use of a 74S381 in the most-significant ALU slot.

The total addition/subtraction logic-delay time for a mill like those of the last several figures goes up only rather slowly as the length of the mill increases — roughly as the *logarithm* (to the base 2) of the length of the mill. For a ripple-carry adder, on the other hand, the total addition/subtraction logic-delay time goes up *linearly* as the length increases. See what's so good about carry-bypass circuits?



NOTE:
'S382 (NOT SHOWN)
PARALLELED TO M.S.
'S381 TO PRODUCE
"OVERFLOW" SIGNAL.
FASTEST C24 IS
AVAILABLE AS SHOWN.

Figure 9. 24-Bit Mill



NOTE: The highest-level $\overline{P}$ and $\overline{G}$ outputs do not include the effects of bits 63-60.

Figure 10. 64-Bit Mill

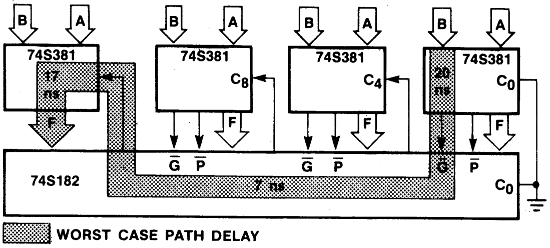


Figure 11. Critical Carry Paths in a Mill

Booth Multiplication

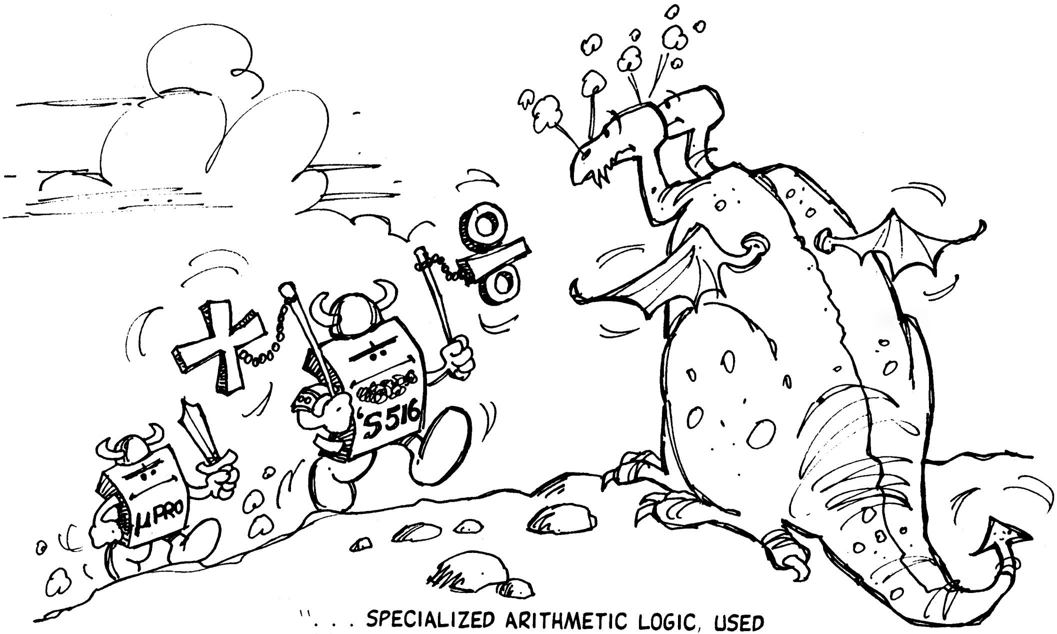
And so we arrive at the promised discussion of Booth multiplication. This is a very different sort of topic, and won't involve any more logic design, since unless you are a professional minicomputer designer (as I used to be) this technique is for you something which is used *within* a chip, rather than *among* several chips as is carry bypassing.

In particular, Booth multiplication is used in two rather interesting chips which my employer again just happens to make, the 'S516 and 'S508 "bus multipliers." These chips can perform multiplication, accumulate-multiplication, and division using 16-bit operands ('S516) or 8-bit operands ('S508). They are intended to function as "co-processors" (helpers) along with 16-bit or 8-bit microprocessors. An 'S516 or 'S508 is "microprogrammed" by getting opcodes one at a time from a microprocessor, except that when it is in the actual computing loop for multiply or divide it finishes the iterative job without needing further opcodes. Because these chips operate at bipolar rather than MOS speeds, they can considerably enhance the effective speed on real-time problems of even large microprocessors.

Reference 3 shows how an 'S516 teamed up with a 16-bit microprocessor can be used for speech synthesis, which is a well-known multiplication-intensive application. There are many such design contexts where specialized arithmetic logic, used together with your microprocessor, can provide extra muscle for handling formidable problems.

Enough crass commercialism — the property of these two chips which is of interest here is that multiplication gets done at the rate of *two* bits per clock, whereas division gets done at the rate of only *one* bit per clock and so takes roughly twice as long. It would seem as if this speed gain must cost a lot of "hardware" (in this case silicon surface area) to achieve for multiplication, right? Wrong. It costs almost nothing. The secret is, of course, Booth multiplication. Keep on reading, and it won't remain a secret for you.

As you probably realize, the most basic way for a computer to multiply two binary numbers together is by repetitive addition. The multiplier is examined one bit at a time, beginning with the least-significant bit. (Well, you *can* begin at the other end, but it is *much* less convenient.) Wherever the multiplier bit is a "1," the multiplicand gets added into the partial product, which then gets shifted to the right by one bit position with respect to the multiplicand. Wherever the multiplier bit is a "0," the shift occurs without the multiplicand ever getting added in, or alternatively you can think of zero getting added in instead of the multiplicand.



". . . SPECIALIZED ARITHMETIC LOGIC, USED TOGETHER WITH YOUR MICROPROCESSOR, CAN PROVIDE EXTRA MUSCLE FOR HANDLING FORMIDABLE PROBLEMS . . ."

One of the earliest, and still one of the best, ways to speed up this whole process is to provide a complete full-length mill, with one set of inputs connected to the multiplicand, for *every* bit position of the multiplier. The other set of inputs is connected to the outputs of the mill immediately above. Either the multiplicand, or else zero, is gated to the multiplicand inputs for that mill according to whether the value of the multiplier bit which corresponds to that particular mill is "1" or "0." The entire humungus arrangement of n n-bit mills, comprising an nxn array of full adders, performs n-bit multiplication on a flow-through basis; the final product is available at the outputs of the bottom-most mill after a "ripple time." This configuration is either called a "Cray multiplier" after its inventor Seymour Cray, or else just a "combinatorial multiplier." It happens that my employer also makes an 8x8 Cray multiplier chip, the 67558, and that the Booth multiplication technique is used within this chip to reduce the number of full adders needed to perform the multiplication by half, although in the newer and faster 'S557 and 'S558 the circuit designer chose to go back to straight Cray multiplication. But I digress again.

The Cray approach is big, fast, simple, clean, and too expensive for some applications. Before LSI semiconductor technology arrived on the scene, it was too expensive for all but a very few applications!

Anyway, the very, very first technique devised by clever designers for speeding up multiplication was to precompute, and keep handy, several multiples of the multiplicand, and select these by examining groups of bits of the multiplier and using that group value to address a small memory or register array where these several multiples of the multiplicand were temporarily kept. This sort of setup is called a "multiple generator." To do 2 bits at a time, the multiple generator needs to contain 1X, 2X, and 3X, where X is the multiplicand.

One obvious economizing move is to provide a second path leading to 1X, so that 1X may be read out left-shifted by one place as well as directly, thereby in effect reading out 2X. A second move is to subtract 1X from the partial product instead of adding it in whenever the addition of 3X is called for, that is, whenever the multiplier group of two bits consists of 11, thus eliminating the need to actually store any multiple of X except 1X. So in two fell swoops we have eliminated the table of multiples of the multiplicand, and yet we are still getting *two* bits of the multiplier, rather than just one, processed on every repeated-addition cycle. But wait a minute! After a cycle in which we have subtracted 1X instead of adding in 3X, it is obvious that the partial product is temporarily not quite correct! So it has to be fixed up by adding in 1 to the next two-bit group examined from the multiplier, which ensures that an extra 1X will get added in on the next cycle. This procedure has exactly the same effect as if an extra 4X got added in on *this* cycle, which would also fix things up since we subtracted 1X instead of adding 3X. Follow all that? Unfortunately, we all of a sudden need another mill, or at least a "half adder" capable of incrementing a number, hooked up directly to the *multiplier* since it may need to be fixed up on each cycle. I have been told that, before Booth multiplication was perfected, big computers were actually built which worked this way, and it *did* cut multiplication time in half.

The Booth algorithm is actually not too big a change from the scheme just described. Conceptually, it is a procedure for making the needed correction at each step without having to propagate a carry all the way up the multiplier, in a way which at the same time may introduce another error which must be fixed up on the next following cycle, and so forth, with everything coming out all right at the very, very end. To multiply using two bits of the multiplier at a time, we actually must look at *three* bits at a time — the current group of two bits, and the most-significant bit in the group which was looked at on the preceding cycle.

And now, where the current group of two bits has value 10, we *subtract* 2X from the partial product instead of adding 2X. (This seems like a step backwards, but don't worry—it will all work out! Now, however, we can tell just by looking at this most-significant bit from the preceding cycle what happened during that cycle — if that bit is "1," we subtracted instead of adding during the preceding cycle and hence our partial product is too small by 4X from *that* cycle and therefore too small by 1X for this cycle, and so we must correct for that on this cycle. The upshot is:

CURRENT GROUP OF TWO BITS	M.S. BIT FROM PREVIOUS GROUP	CURRENT MULTIPLICAND MULTIPLE TO BE ADDED IN	CORRECTION MULTIPLICAND MULTIPLE TO BE ADDED IN	NET MULTIPLICAND MULTIPLE TO BE ADDED IN
0 0	0	0X	0X	0X
0 0	1	0X	1X	1X
0 1	0	1X	0X	1X
0 1	1	1X	1X	2X
1 0	0	-2X	0X	-2X
1 0	1	-2X	1X	-1X
1 1	0	-1X	0X	-1X
1 1	1	-1X	1X	0X

Table 6. Two-Bits-at-a-Time Booth Multiplication

As you can by now see, this process is self-correcting as it goes along. All that is necessary to start it off properly is to force the leftover most-significant bit from the previous group to zero at the very beginning, since there actually wasn't any previous group. This process also *ends* properly as long as the most-significant two bits of the multiplier are either 00 or 01, since there is no borrowing against the in-this-case-nonexistent next group of two multiplier bits for those values. But, curiously enough, if the most-significant two bits of the multiplier are 10 or 11, and the most-significant bit of these is interpreted as the sign bit of the entire multiplier, and the multiplier and multiplicand follow the "twos-complement" convention for negative numbers whenever they aren't positive, then the final product is *also* correct! Proving this fact is wildly outside the scope of this paper, but it is proved in an admirably clear three-decades-old paper by Mr. Booth himself, for the case of what we would today have to call *one*-bit-at-a-time Booth multiplication!! (See Reference (5).)

I developed Booth multiplication for the two-bits-at-a-time case because in some ways it is intuitively the easiest case, it is justifiable on speed-improvement grounds, and besides it is the way that the 'S516 and 'S508 work. However, the basic Booth-multiplication principle works for *any* number of bits at a time. The reason why Booth devised the algorithm in the first place was *not* speed improvement, but to eliminate a rather messy cleanup phase at the end of a multiply operation when two twos-complement numbers are multiplied together. It was only much later that this basic algorithm got wedded to table-lookup multiplication as a way of simplifying (or, in the case you have just seen, essentially eliminating) the table. The two-bits-at-a-time algorithm was, as far as I know, first published in 1975 by Rubinfield (Reference (6)).

The *three*-bits-at-a-time algorithm, which is the final curve-ball I'm going to throw at you in this paper, hasn't to my knowledge been published prior to right now. It has been *used* quite a bit in minicomputers for about a decade, but minicomputer manufacturers are a secretive lot and don't let their designers publish proprietary tricks of this sort for their competitors' designers to read all about. To be sure, fairly similar algorithms for two- and three-bits-at-a-time multiplications *were* published back in 1961 by MacSorley (see Reference (7)), but these algorithms aren't exactly Booth multiplication despite some statements which you may now and then read to the contrary. The key difference is that MacSorley's algorithm looks at the least-significant bit in the *next* oncoming group of bits instead of the most-significant bit in the group just gone by. After this algorithm gets underway it is much like Booth multiplication; but, for reasons I haven't space for here, there are more and dirtier inherent complications both in starting off the multiplication process and in ending it.

The reasoning applicable to the development of the table for three-bits-at-a-time Booth multiplication is identical with that for the two-bits case, so here's the table:

CURRENT GROUP OF THREE BITS	M.S. BIT FROM PREVIOUS GROUP	CURRENT MULTIPLICAND MULTIPLE TO BE ADDED IN	CORRECTION MULTIPLICAND MULTIPLE TO BE ADDED IN	NET MULTIPLICAND MULTIPLE TO BE ADDED IN
0 0 0	0	0X	0X	0X
0 0 0	1	0X	1X	1X
0 0 1	0	1X	0X	1X
0 0 1	1	1X	1X	2X
0 1 0	0	2X	0X	2X
0 1 0	1	2X	1X	3X
0 1 1	0	3X	0X	3X
0 1 1	1	3X	1X	4X
1 0 0	0	-4X	0X	-4X
1 0 0	1	-4X	1X	-3X
1 0 1	0	-3X	0X	-3X
1 0 1	1	-3X	1X	-2X
1 1 0	0	-2X	0X	-2X
1 1 0	1	-2X	1X	-1X
1 1 1	0	-1X	0X	-1X
1 1 1	1	-1X	1X	0X

Table 7. Three-Bits-at-a-Time Booth Multiplication

After comparing Table 6 with Table 7, you should be able to figure out the pattern and come up with your own n-bits-at-a-time Booth-multiplication table for *any* value of n, subject to writer's cramp as n gets large. The three-bits case is, however, probably the highest-order one with compelling practical advantages in designing high-speed arithmetic systems; only one special multiple (3X) of the multiplicand actually needs to be computed at the outset of the algorithm and stored, since 2X, 4X, and the negatives of all of these can be obtained by shifting and complementation. When one goes to the four-bits case, the number of added new setup steps almost wipes out the additional number of multiplication cycles saved, and quite a bit of extra hardware is needed, which is *not* the situation when one goes from the two-bits case to the three-bits case at least if the latter is implemented cleverly. If you need even more speed than you can get even using three bits at a time, probably you better start looking at Cray multiplication and 'S558s.

Wrapup and Homage

To those of you are still with me after the preceding mental exercises, congratulations! You now understand some techniques which at least *used* to be considered "insider" tricks in digital systems design, a decade or so ago. If you are by now intrigued with digital arithmetic techniques and wish to dig much deeper, consult references (7), (8), (9), (10), and (11). If you want to actually *build* something and make it work, you'll need some data books from semiconductor manufacturers for more complete information; references (12), (13), and (14) are a good basic starter set.

Since portions of this paper got written on company time, and all of it got written using a company-paid-for time-sharing-system editing facility, I owe Monolithic Memories — and particularly my boss, Shlomo Waser — a vote of thanks for blessing this project and footing the bill.

Most of the development given in this paper of Booth multiplication I owe to a friend and (as of that time) fellow Florida minicomputer designer, Rick Johnston. Rick is now at General Electric in upstate New York.

References

(1) "Using LSI to Crunch Numbers at High Speed: an Overview," C. Norman Winningstad, *1979 Wescon Professional Program*, Session 18 reprint, paper number 18/1. Available from Wescon/Electronic Conventions, Inc., 999 North Sepulveda Boulevard, El Segundo, California 90245.

(2) "Bus Oriented Multiplier/Divider — Support Chip for the 16-bit Microprocessors," Shlomo Waser, *1978 Wescon Professional Program*, Session 25 reprint, paper number 25/4. Available from Wescon/Electronic Conventions, Inc., 999 North Sepulveda Boulevard, El Segundo, California 90245.

(3) "Medium-Speed Multipliers Trim Cost, Shrink Bandwidth in Speech Transmission," Shlomo Waser and Dr. Allen Peterson, *Electronic Design*, 2/1/1979, pages 58-65.

(4) "Mixed Logic; A Tool for Design Simplification," Paul M. Kintner, *Computer Design*, 8/1971, pages 55-60.

(5) "A Signed Binary Multiplication Technique," Andrew D. Booth, *Quarterly Journal of Mechanics and Applied Mathematics* (British), Volume IV Part 2 (1951), pages 236-240.

(6) "A Proof of the Modified Booth's Algorithm for Multiplication," Louis P. Rubinfield, *IEEE Transactions on Computers*, 10/1975, pages 1014-1015.

(7) "High-Speed Arithmetic in Binary Computers," O. L. MacSorley, *Proceedings of the IRE* (now IEEE), 1/1961, pages 67-91.

(8) *The Logic of Computer Arithmetic*, Ivan Flores, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1963.

(9) *Computer Arithmetic: Principles, Architecture and Design*, Kai Hwang, John Wiley & Sons, Inc., New York, New York, 1979.

(10) *An Introduction to Arithmetic for Digital System Designers*, S. Waser and M. J. Flynn, Holt, Rinehart & Winston, Inc., New York, 1982.

(11) "Big, Fast, and Simple — Algorithms, Architecture, and Components for High-End Superminis," Ehud Gordon and Chuck Hastings, *1982 Southcon Professional Program*, Session 21 reprint, paper number 21/3. Available from Southcon/Electronic Conventions, Inc., 999 North Sepulveda Boulevard, El Segundo, California 90245 or as Application Note AN-111 from Monolithic Memories, Inc., 1165 East Arques Avenue, Sunnyvale, California 94086.

(12) *Bipolar Microprocessor Logic and Interface Data Book*, Advanced Micro Devices, Inc., 901 Thompson Place, Sunnyvale, California 94086.

(13) *Bipolar LSI 1982 Data Book*, Monolithic Memories, Inc., 1165 East Arques Avenue, Sunnyvale, California 94086.

(14) *The TTL Data Book for Design Engineers*, Texas Instruments, Inc., Post office Box 5012, Dallas, Texas 75222; second edition, third printing, dated 1981. Here there is a supplement with the imaginative title *1981 Supplement to the TTL Data Book for Design Engineers*, second edition.

Minimum Chip-Count Number Cruncher Uses Bipolar Co-Processor*

C. Hastings, E. Gordon, and R. Blasco

The high speed, programmability, and flexibility of bipolar parts are exploited in a floating-point arithmetic co-processor board which is presented in this paper. The operation of the co-processor and the detailed implementation is supplied. This paper concludes with a comparison of the performance of the bipolar co-processor with other implementations. This design is found to have much better performance while maintaining a low chip count, thus providing a cost-effective solution.

*This paper is a slightly updated version of the paper by the same name which appeared in the the *Wescon/81 Professional Program Session Record,* Session 3 reprint, paper 3/1; 15-17 September 1981.

Minimum Chip-Count Number Cruncher Uses Bipolar Co-processor

C. Hastings, E. Gordon, and R. Blasco

Introduction

Bipolar components can be used to construct a high-performance number cruncher that provides a cost-effective alternative to MOS devices in microprocessor systems. The high speed, programmability, and flexibility of the bipolar parts are exploited in a floating-point arithmetic co-processor board.

The high-speed bipolar components provide a floating-point arithmetic capability with more than twice the throughput of the fastest 16-bit MOS co-processor arrangement currently available.

Programmable components not only permit implementation of a wide variety of arithmetic operations, but also provide true co-processor operation with virtually any microprocessor. Eight-bit microprocessor systems can be upgraded to provide arithmetic throughput that rivals the best sixteen-bit machines while preserving the utility of existing software. The bipolar co-processor can also be used to improve the throughput of new 16-bit systems.

The flexibility of the bipolar approach allows the co-processor to be implemented with approximately 30 packages, including all overhead for system bus interface. Performance and package count can be traded off to optimize the co-processor for any system requirement. The design can be adapted for interface with any of the popular system buses (MULTIBUS™, STD™, IEEE-696 (S-100), etc.) with minimal modification.

Co-processor Operation

A co-processor monitors data fetched by the "host" microprocessor, and recognizes instructions and data intended for itself. This arrangement is much faster than routing all data and commands through the host. Information is picked off of the system bus directly, avoiding extra host "write" cycles. All of the program branch and loop operations and all of the address modes of the host processor are available to the co-processor, since all addressing is provided by the host.

The co-processor occupies an I/O port or a memory-mapped address in the system. The host activates the co-processor by writing a command to the appropriate port or address. This command sets up the co-processor for the operations and data formats to be handled. Once activated, the co-processor recognizes specified host instructions and retrieves its input data from the bus. The host reads the appropriate port or address to fetch the results of the arithmetic operation. If the co-processor is still busy, a bus status signal will be activated to place the host in a "wait" state until the output data is ready.

Figure 1 illustrates the data flow during an input fetch operation. A programmable array logic (PAL®) device is used to decode the selected host "fetch" instruction. The code for the selected instruction can be customized for any host by re-programming this PAL.

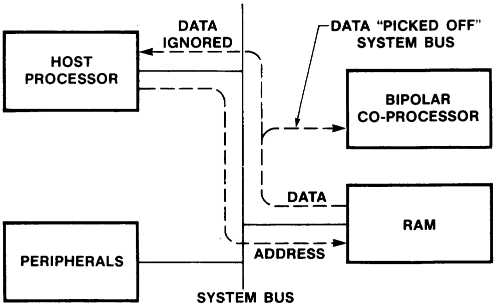


Figure 1. Co-Processor Input Data Flow

For best efficiency, a 16-bit register load instruction should be used. Inputs are loaded both into the co-processor, where they are used, and into the selected host register, where they are ignored. If this register is required for host processing in parallel with the arithmetic operation, then it should be "pushed" and "popped" from the host stack area to preserve its contents. Normally this overhead can be avoided by careful programming.

The co-processor monitors the system bus for interrupt or direct memory access (DMA) operations. Occurence of the selected "fetch" instruction during these operations is ignored by the co-processor. This prevents false data from being loaded into the co-processor while execution of the main host program is suspended.

All instructions used are part of the normal host instruction set. Software to operate the co-processor is readily created using a standard macro-assembler for that host. Macros are created to perform the necessary setup and fetch operations for the desired arithmetic function. These macros are then used in the main assembly source program, as desired. Existing software is easily modified to utilize the co-processor capabilities by replacing software arithmetic routines with the appropriate macro command sequences.

Ready/wait handshaking is used during retrieval of the co-processor output. This approach is normally much faster than interrupting the host upon completion of an operation.

The co-processor is functionally a bus "slave" device. To maintain synchronization and to follow the operation of the host, however, to co-processor must respond as a bus "master" to certain bus states. For example, the bus "ready" line must be monitored for the availability of stable data from the system memory. The co-processor will enter "wait" states, just like the bus master, until the data is valid. The co-processor must interpret special formatting signals, such as 8-bit versus 16-bit data transfers, in the same way that the host will interpret those signals.

Number-Crunching Functions

The co-processor uses an internal microprogrammed controller to implement various number-crunching functions. Functions may be added or modified to suit users' needs by changing the microcode, which is stored in bipolar PROMs.

The bipolar co-processor can perform the following arithmetic operations:

- **Floating-point add, subtract, multiply, divide, and compare**
- **Fixed-point add, subtract, multiply, divide, and compare for 16- or 32-bit integers**
- **Fixed-to-floating-point conversion**
- **Normalize (left-justify) inputs and outputs**
- **Push or pop numbers from an internal stack**
- **Transcendental functions (sine, cosine, tangent, log, square root, etc.)**
- **Input/output format conversion**

Floating-point calculations are performed with an 8-bit biased exponent (127 is added to the true exponent value) and a 30-bit mantissa. The proposed IEEE floating-point standard specifies a 23-bit mantissa in sign-magnitude form. A comparison of the IEEE and internal word formats is given in Figure 2. Internal numbers are in two's-complement form for compatibility with the bipolar multiplier used in the co-processor. Seven guard bits are provided to ensure the accuracy of complex operations. Input and output of the co-processor board is normally in the internal format of Figure 2(b), with conversion to the IEEE format of Figure 2(a) available as an optional command.

Internal numbers are stored in fractional form, with the binary point to the left of the mantissa MSB. Fractional storage eliminates mantissa overflow on multiply. Numbers are pre-scaled to prevent overflow on add and subtract. Special procedures to eliminate overflow on divide are described in the next section. Mantissa overflow is thus completely avoided. Results are always re-normalized before output.

Errors can occur due to exponent overflow, division by zero, or transcendental arguments out of range. The co-processor will activate the system bus error line to flag the occurrence of a processing error.

Implementation

An implementation of the bipolar co-processor for the IEEE S-100 bus is shown in Figure 3. The co-processor architecture consists of several distinct circuits interfaced to a common 16-bit internal data bus:

- **Mantissa processor**
- **Exponent processor**
- **RAM**
- **MSB position detector**
- **Bit/byte manipulator**
- **Control sequencer**
- **Instruction ROM**
- **Constant (immediate) data**
- **Lookup table**
- **Control decode logic**
- **Bus interface**

Mantissa Processor

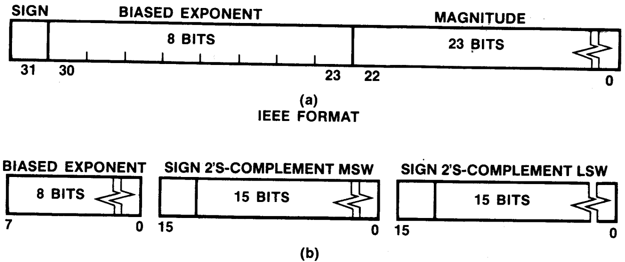
Mantissa arithmetic is performed using the Monolithic Memories 74S516 multiplier/accumulator chip. This part performs multiplication and accumulation of 16-bit signed numbers in about 1.25 microseconds. Division of a 32-bit signed number by a 16-bit signed number can be performed in 2.5 microseconds.

These numbers are based on the 8-MHz "typical" operating frequency; the "worst-case" operating frequency is 6 MHz over the commercial temperature range, and is 5 MHz over the

While faster bipolar multipliers are available, the functional flexibility and bus orientation of the 74S516 make this part ideal for use in a minimum-parts-count co-processor.

The 74S516 speed is well matched to the speed of other co-processor operations performed in parallel with the 74S516 operations. As a result, throughput is not severely penalized by the use of this part.

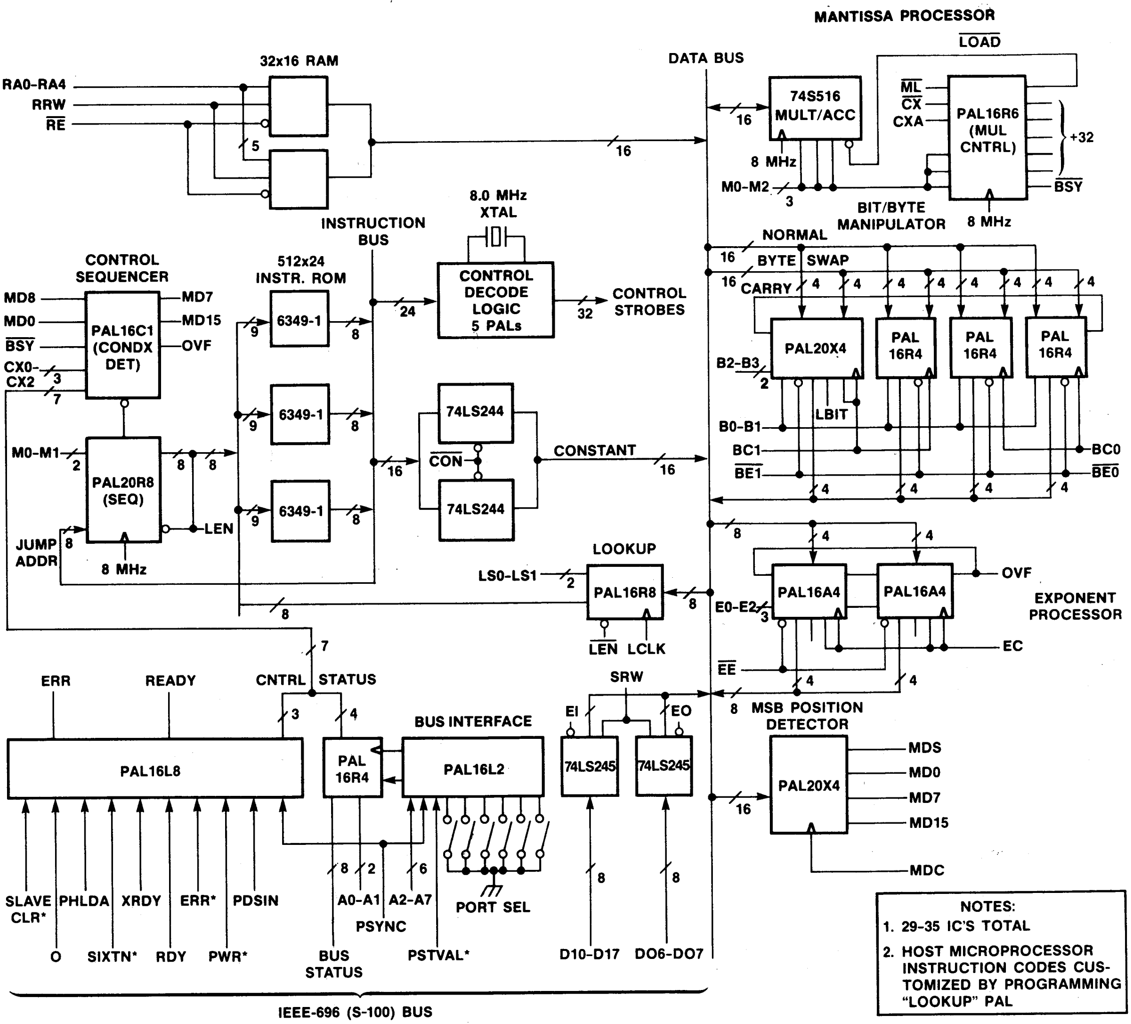
The 74S516 has special instructions to accommodate the handling of double-precision multiplications and accumulations. The internal word format of the co-processor is directly compatible with these double-precision operations. A mantissa precision of 30 bits is directly handled by the 74S516, with only a speed penalty involved.



(a)
IEEE FORMAT

(b)
CO-PROCESSOR INTERNAL FORMAT

Figure 2. Floating-Point Word Formats



NOTES:
1. 29-35 IC'S TOTAL
2. HOST MICROPROCESSOR INSTRUCTION CODES CUSTOMIZED BY PROGRAMMING "LOOKUP" PAL

Figure 3. IEEE-696 (S-100) Bus Implementation of Bipolar Co-processor

Double-precision division is more complicated. First, the inverse of a fraction is always greater than 1, forcing a violation of the internal number system. Second, the 74S516 cannot perform a double-precision division directly.

Division of X/Y can be accomplished by calculating 1/Y and then multiplying the result by X. To keep all numbers within the fractional number system, 1/2Y is actually calculated. If Y is normalized (between 0.5 and 1), then 1/2Y is always between 0.5 and 1, ensuring closure of the number system under this operation. This result is multiplied by X and normalized, taking account of the required scale factor correction.

Higher precision than that provided directly by the 74S516 is achieved using a recursion formula. We seek the root of the equation:

$$\frac{1}{q} - 2Y = 0 \qquad (1)$$

The Newton-Raphson root-solving method provides the following recursive formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (2)$$

Calculating the derivative of (1) and substituting into (2) yields (after some manipulation):

$$q_{n+1} = 2q_n (1-Yq_n) \qquad (3)$$

If Equation (3) is calculated properly, intermediate results will always be within the fractional number system if Y is normalized.

Error analysis of the efficiency of this recursion formula is performed by substituting:

$$q_n = \frac{1}{2Y} + e \tag{4}$$

into (3) and manipulating, yielding:

$$q_{n+1} = \frac{1}{2Y} - 2Ye^2 \tag{5}$$

Equation (5) indicates that each iteration of (3) will produce 2N-1 significant bits, where N is the precision of the input estimate. The 74S516 is used to create the initial estimate of q. This estimate will contain 15 significant bits. A single pass through (3) will then provide the true quotient to a 29-bit precision.

In order to simplify the microprogram, a PAL device generates a busy/ready signal for the mantissa processor. This eliminates wasteful "NO-OP" instructions in the microcode.

Exponent Processor

Two PAL16A4 devices implement the exponent processor. A feature of this circuit is a latched carry bit, which permits chained operations (with carry) of arbitrary multiples of 8-bits. The latched carry bit also functions as an exponent overflow flag at the completion of an operation. Three control lines specify eight possible operations:

CODE	OPERATION
000	Clear accumulator
001	Load A
010	A .AND. B
011	A .XOR. B
100	A + B
101	A - B
110	B - A
111	Compare A and B

NO-OPs are provided by gating the processor output clock.

Operations 011 through 111 may be chained by using the latched carry feature. The exponent processor accepts only one 8-bit input at a time. Chained operations are terminated by a clear command.

The compared operating forces the output of the processor to states that can be easily decoded by the MSB position detector. If A is less than B, the output of a chained operation forces the sign bit = 1. If A equals B, the output is all zeroes. If A is greater than B, the processor forces the sign bit = 0 and all significant bits = 1. The latched carry bit is used to remember the results of previous comparisons in the chain.

RAM

A 32x16-bit random access memory (RAM) stores intermediate values and provides a stack area for the co-processor. This RAM requires two to eight packages, depending on the bipolar parts used.

MSB Position Detector

A PAL20X4 locates the MSB position of any word present on the internal 16-bit data bus. The position detector works with positive or negative two's-complement numbers. Outputs indicate the sign and whether the number is greater than or equal to 2^{-1}, 2^{-7}, or 2^{-15} in absolute value.

The position detector has a registered output. The register clock is gated, so that the detector may remember the MSB position of previous bus words. The control microprogram can branch conditionally on the outputs of the position detector.

Bit/Byte Manipulator

Three PAL16R4s and one PAL20X4 implement a bit/byte manipulator circuit. This special circuit permits rapid adjustment of word formats. It is used for input/output format conversion, normalization, and other special operations.

All four PALs respond to two control bits, which specify the following operations:

CODE	OPERATION
00	Normal load of 16-bit word
01	Load word with bytes interchanged
10	Shift left
11	Shift right

In addition, the PAL20X4 implements a bidirectional latched carry bit that is used to modify single bits within a word, or to remember carries for shift left/shift right operations of 32-bit words. This latched bit can be used to invert the sign bit of the word, implementing a (1-X) function useful in calculating Equation (3). It can also be used to implement a "sticky bit" for full compatibility with the MOS arithmetic processor internal formats. The PAL20X4 responds to two additional control lines, which specify the operation of the latched bit as follows:

CODE	OPERATION
00	Hold previous bit
01	Save carry bit
10	Save sign bit
11	Save complement of sign bit

The bit/byte manipulator circuit provides the operations required for flexible co-processor internal/external word format compatibility, without burdening the arithmetic processors with these tasks.

Control Sequencer

Two PALs implement a microprogrammed control sequencer for the co-processor. A PAL20R8 serves as a program counter, while a PAL16C1 functions as a branch condition detector.

The program counter can address 256 program steps. Two control lines set up conditional operations, as follows:

CODE	OPERATION
00	Increment
01	Skip (increment by two)
10	Branch to jump address
11	Repeat (no increment)

The operations are conditioned on the output of the condition detector. If the required condition is not met, the program counter defaults to operation 00 (increment).

The condition detector monitors flags and status bits generated within the co-processor, and generates a condition output signal. This output may be used for conditional program branches, or for conditional mantissa processing. The latter is accomplished by gating the LOAD output of the multiplier control PAL with the condition output.

Instruction ROM

Three Monolithic Memories 6349-1 PROMs form a 512x24-bit instruction memory. This memory stores the microprogram for the co-processor. In addition, a second memory page is available for lookup- and jump-tables. All operations of the co-processor are determined by the microcode stored in this memory. The use of PROMs permits easy microprogram modification.

Constant (Immediate) Data

Two buffers gate 16 bits of the instruction memory word on to the internal data bus. Constants and "immediate" operands may be stored in the instruction ROM and accessed by enabling these buffers.

Lookup Table

A PAL16R8 assembles a lookup table address. The address is assembled from data present on the internal data bus, and two "table select" lines. The assembly algorithm is programmable in this PAL, permitting easy modification of table start addresses. The lookup enable line (LEN) provides the ninth address bit to the instruction memory, automatically selecting the lookup table page.

Data lookup tables are useful in calculating transcendental functions. Table lookup plus interpolation is a very efficient method of computing most of these functions. If the lookup tables are large enough, full precision results can be obtained faster with this technique than can be obtained using power series or CORDIC algorithms.

Vectored jump tables can also be implemented with the lookup feature. This capability is used to interpret setup commands from the host microprocessor. In addition to a selected register load instruction, the host interrupt "return" instruction must be interpreted to re-activate the co-processor following a host interrupt. The PAL program is customized to the host instruction codes. The co-processor is "personalized" for any host by re-programming this one PAL.

Control Decode Logic

Approximately five PALs are required to interpret the instruction word and generate glitch-free strobes to operate the co-processor circuit.

Spare I/O pins on one of the PALs can be used to implement a simple crystal oscillator circuit. This crystal oscillator drives all of the clocked co-processor circuits. The 8-MHz "typical" clock frequency translates to a 125-nanosecond basic cycle for the co-processor.

Bus Interface

A full IEEE-696 (S-100) bus interface is shown in Figure 3. This circuit would be modified for interface to other bus formats.

All relevant bus status and control lines are handled by the bus interface. Either 8- or 16-bit word length is supported. Host interrupts and DMA transfer are recognized. Necessary gating and strobe signals are handled to ensure that bus information is sampled only when valid. Not taking shortcuts here ensures reliable interface with the bus.

A PAL16L2 decodes the I/O port address, determined by a DIP switch. PSYNC and PSTVAL* are used to create a strobe to reliably sample bus status.

Bus status is sampled by a PAL16R4 and compressed to a four-bit code for use by the condition detector. All eight status signals and the two low-order address bits are sampled. The address bits are used when multi-byte transfer of 16-bit data is used by the host.

Two 74LS245 buffers are arranged to interface either 8- or 16-bit data transfers in accordance with the IEEE standard. The bit/byte manipulator circuit is used to assemble and disassemble co-processor data for interface to the S-100 bus.

Bus control information is compressed to a 3-bit code by a PAL16L8. The SIXTN* and RDY lines are properly sampled by the master clock (phi) line and PSYNC.

The ERR* and RDY lines are bi-directional, allowing output of this information to the host from the co-processor. Tri-state output enables are individually programmable on the PAL16L8, permitting simulation of the required open-collector format of these signals.

Performance

The performance of the bipolar co-processor is compared to that of two advanced MOS arithmetic processor chips in Table 1. All entries in this table assume normalization of the output mantissa. The minimum to maximum variations are primarily due to normalization time.

It can be seen from the table that the bipolar co-processor affords a speed advantage of 3.6 to 13.3 over the AMD 9512-1, and 1.7 to 3.4 over the Intel 8087. The Intel part represents the best 16-bit numeric co-processor currently available (it is used with the Intel 8086 or 8088 microprocessors). Despite the high-performance MOS technology used in these products, bipolar components still have a performance edge.

Table 1.
Comparison of Arithmetic Processor Performance

OPERATION	AMD 9512-1		INTEL 8087		BIPOLAR	
	min	max	min	max	min	max
Floating-point add	18.6	163.8	18	24	5.8	12.3
Floating-point subtract	17.9	163.8	18	24	5.8	12.3
Floating-point multiply	61.5	81.3	22	25	6.4	7.3
Floating-point divide	73.0	84.5	43	45	22.5	26.0

NOTES:

① All table entries are in microseconds.

② All operations include normalization of output.

③ Input/output access times not included.

④ The "typical" clock frequency of 8 MHz is assumed for 'S516 operation.

The total package count of 30 allows the co-processor to be laid out on just one board. The package count compares very favorably with a bipolar bit-slice implementation, reflecting the effective use of PALs in the control circuits.

The component cost of the bipolar co-processor is comparable to the current cost of the MOS devices. Production cost can be reduced by replacing the PALs and PROMs with available mask-programmable devices.

The flexibility of the bipolar approach allows performance, package count, and cost to be traded off to suit the user's needs. Monolithic Memories 74S557/8 multipliers can be used to substantially improve mantissa-processing speed, at the cost of 10 to 15 packages. Providing a 32-bit wide bit/byte manipulator (using a total of 8 PALs) would dramatically reduce the time required for normalization. An on-board microprocessor, such as a Motorola 6809 or Signetics 8X300, could reduce package count at the expense of throughput.

A DMA controller might be added to the basic co-processor circuit. When used in conjunction with a larger on-board RAM capacity, efficient vector and array calculations could be performed.

The bipolar co-processor thus represents an efficient, flexible, and cost-effective means of upgrading the number-crunching capability of microprocessor systems.

References

1. *PAL Applications Handbook*, available from Monolithic Memories, Inc.
2. S. Waser, "Designing Your Own Floating-Point Processor," paper presented at the EE Times "Advanced Concepts in Using Microprocessors" Seminar, June 18–19, 1979.
3. K. Hwang, *Computer Arithmetic*, John Wiley and Sons, New York, 1979.
4. K. Elmquist, et al., "Standard Specification for S-100 Bus Interface Devices," *Computer Magazine*, July, 1979, p. 28 ff.
5. S. Waser and M.J. Flynn, *An introduction to Arithmetic for Digital System Designers,* Holt, Rinehart & Winston, New York, 1982.

Notes

Supercharging Microprocessor Arithmetic

Suneel Rajpal

Abstract

MOS or bipolar microprocessors have fairly extensive instruction sets. However applications requiring high-speed arithmetic computations such as multiply and divide operations, may preclude the use of the microprocessors.In situations where extensive number crunching is required with minimal external hardware, low cost, and a short development cycle, the Monolithic Memories SN54/74S516, 16-bit multiplier/divider, provides an excellent solution.

This paper presents simple hardware interfaces for using the 'S516 with the Am29116 in a graphics environment. Another application using the 'S516 with the 8086 is also discussed. The performance of these microprocessors with and without the 'S516 are tabulated and the speed enhancements achieved are 6:1 for the Am29116 and 3:1 for the 8086, for a multiply operation.

* This paper is a slightly updated version of the paper by the same name which appeared in the Northcon/83 Professional Program Session Record, Session 16 reprint, paper 16/4, 10-12 May 1983.

Summary

Specialized arithmetic logic, used together with bipolar or MOS microprocessors, can provide extra muscle for handling extensive number-crunching operations. The Monolithic Memories SN54/74S516 bipolar multiplier/divider/accumulator can team up with popular 16-bit or 8-bit microprocessors to significantly improve the arithmetic throughput.

The S516 can be used with both LSI-microprocessor-based and horizontal-microcode-driven processors. For instance, by adding minimal external circuitry the multiply performance for the Advanced Micro Devices Am29116 bipolar microprocessor can be enhanced by about 6:1; and that of the Intel 8086 can be enhanced by about 3:1.

This paper presents simple hardware interfaces for using the S516 with the above mentioned microprocessors. The performance of the multiplier/divider under various conditions is tabulated and plausible areas of application are discussed.

S516 Operation

The S516 is the only 16-bit bipolar multiplier/divider available today. It comes in a 600-mil 24-pin package. Special hardware and Booth Algorithm techniques permit a 10-cycle multiple operation, and a 22-cycle divide operation. The internal architecture of the S516 is shown in figure 1.

The S516 contains four 16-bit registers. Y is the multiplier register; X is the multiplicand/divisor register; W is the least significant half of the double-length accumulator; it holds the resulting product for multiply operations and the dividend for divide operations. Z is the most significant half of the double-length accumulator; it holds the resulting product for multiply operations, and the dividend for divide operations. Z and W can also be used to hold the partial product of a previous multiply cycle, which allow chained multiply operations for wordlengths greater than 16 bits.

To understand the S516, it is important to realize that the state counter drives a PLA decode circuit and this provides controls to the other elements shown in figure 1. To use the S516, the designer does not have to understand the PLA decode or any complex circuit. Rather, the logic external to the S516 must present instruction codes and load the operands, and then the hardware cranks out the results which then may be read out. The state of the S516 is a function of its previous state and the current instruction code. The state transition diagram is shown in Figure 2.

State 0, 8, and 10 represent read states. Repetitive read instructions (Instruction 7) alternate access between the Z(MSW) and W(LSW) output registers. State 0 is the "idle" state, state 8 is at the completion of a multiply, and state 10 is the completion of a divide. Instruction 5 rounds the double-precision result Z,W to 16 bits, with register W set to zero. In state 10, Z holds the quotient and W holds the remainder. In state 8, Z and W hold the most and least significant halves of the product respectively.

The example shown in Table 1, is a step-by-step multiply operation, X∗Y.

$\overline{GO}$	0	0	X	X	0	0
INPUT INSTRUCTION	6	0	X	X	7	7
S516 STATE	0	2	4	12*	8	0
BUS ACTIVITY	X INPUT	Y INPUT	—	—	Z OUTPUT	W OUTPUT

*The 'S516 remains in State 12 for 7 Cycles.

Table 1. The Instruction Sequence for a Multiply Operation

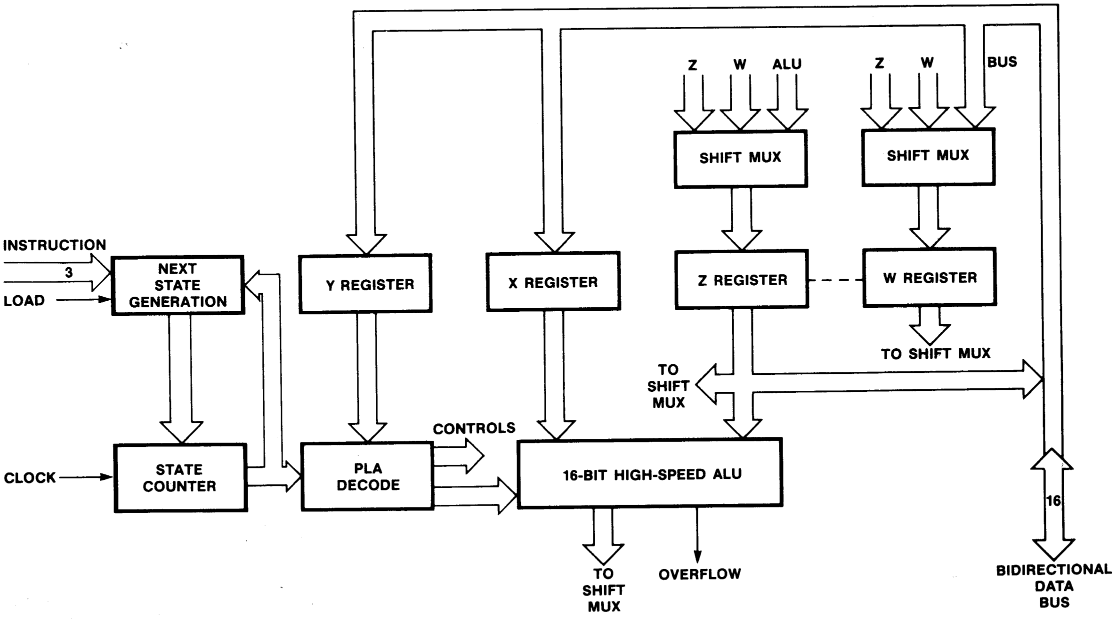
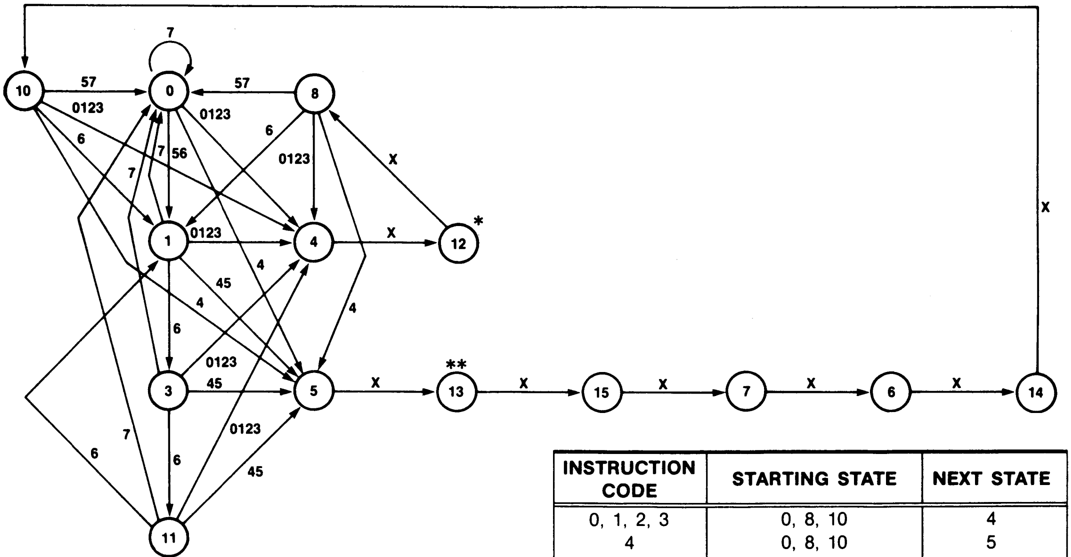


Figure 1. Internal Architecture of the 'S516



INSTRUCTION CODE	STARTING STATE	NEXT STATE
0, 1, 2, 3	0, 8, 10	4
4	0, 8, 10	5
5	0	1
5, 7	8, 10	0
6	0, 8, 10	1
7	0, 8, 10	0

*Loop 7 times for multiplication.
**Loop 14 times for fractional di\ision, or 15 times for integer division.

Figure 2. Transition Diagram for the 'S516 Multiplier/Divider

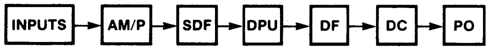
Details of the instruction set are in the SN54/74S516 Multiplier/Divider data sheet. Many instructions are available, including, 28 multiply operations and 13 divide operations. Positive and negative multiply, multiply-and-accumulate, multiply by a constant, single and double length divide, divide by a previously generated result and divide by a constant. The S516 can handle fractions or integers, and uses the twos-complement number system.

The S516/Am29116 Interface

The Am29116 is a versatile 16-bit microprocessor. It has an architecture and instruction set optimized for intelligent controllers in microprogrammed environments. It has 32 working registers, a 16-bit barrel shifter, and bit-manipulation capabilities. However, for multiply and divide the part is relatively slower, since these must be programmed by repetitive add, repetitive subtract, and test conditions respectively. In applications that require higher throughput for multiply and divide operations with a low overhead in dollars and board area, the S516 provides an excellent solution.

Graphics is an excellent potential application area for high-performance controllers such as the Am29116. This paper emphasizes the higher-performance applications within the overall graphics area.

The architecture for a high-performance system is shown in Figure 3. Data is received by an input handler, and it undergoes program-specific processing for each particular graphics application. The output of the input section is generally an abstract structured representation of the images which have to be displayed. The next processing element, the display processing unit (DPU) manipulates this structured representation of the images, by various transformation operations such as rotation, translation, and scaling. In addition to these transformations the DPU may have to perform operations such as "clipping." The output of the DPU is a display file, which contains graphical primitives describing the object produced after modelling and viewing. Finally the graphic primitives map the display file to the screen. For interactive graphics, all the involved processes must be as fast as possible.



AM/P	APPLICATION MODEL/PROGRAM
SDF	STRUCTURED DISPLAY FILE
DPU	DISPLAY PROCESSING UNIT
DF	DISPLAY FILE
DC	DISPLAY CONTROLLER
PD	PICTURE DISPLAY

Figure 3. A High-performance Graphics System Data Flow

In certain areas of the data flow, extensive number-crunching is required. In the DPU, typical transformations involve matrix multiplication for every point in an object's display list. A two-dimensional display list requires multiplying a three-element vector by a 3x3 matrix. Some of the elements in the matrix are always zero but the operations require up to 4 multiply and add operations. In three-dimensional display lists each point requires that a four-element vector be multiplied by a 4x4 matrix, so that 13 multiply and 9 add operations are implied for each point.

Each such display can contain as much as thousands of lines, each of which requires 2 operations per point. Therefore, it is necessary to have high-speed matrix multiplication.

Another consideration is the actual screen representation of data. The process of converting a line, point and area representation to the pixel area of the image storage is called "scan conversion."

A popular form of display is the raster-scan bit-mapped display, where the image is an X,Y matrix of picture elements or pixels which are stored in the display memory or frame buffer. For example, a 1024x1024 display memory can be represented as a 64Kx16 bit-map memory. The image is refreshed by traversing the screen one raster line at a time, by turning the appropriate pixel on the display. However, to determine if a particular pixel has to be turned on requires some calculations. For example, when drawing a line given the endpoints the following operations are performed; computing the slope (a divide operation), and then incrementing the x coordinate by one unit, and then computing the y coordinate which would be the previous value of y plus the slope. The pixel corresponding to the new value of X,Y is then set to a particular value.

The foregoing is the basic incremental algorithm. There exist many algorithms for scan converting lines, circles, and areas listed in (1). These have to be executed each time some or all of the displayed image changes.

The algorithms sometimes contain multiply and divide operations, which may be a bottleneck in updating the picture. Even without multiply and divide routines scan conversions can be a major bottleneck in updating the picture. If the Am29116 is used in graphic applications, then coupled with the S516, it can have significantly better arithmetic throughput for multiply, multiply-and-accumulate, and divide operations.

The above is just one example; the following interface for the S516 can be used in any microcode-driven system. In this example the Am29116 is connected to the S516 using two 24-pin LS546s register transceiver parts. The LS546 is equivalent to two LS374s in a single package, interconnected in a back-to-back manner. Each part has two 8-bit registers, and each of these has their own clock, clock-enable and output-enable control inputs. The 32mA output drive makes the LS546 suitable for bus operations. The clock to the S516 is skewed to meet hold and setup time requirements of the S516. The skew can be created by using a delay line or by using a 20-pin LS344. The interface diagram is given in Figure 4.

The Am29116 transmits the operands over its Y bus and for this application a MOVE operation for the Am29116 takes data from the RAM and places it on the Y bus. Figure 4 has a simplified version of the Am29116 as it applies to this situation. The Am29116 transmits data to the LS546, and then the S516 is loaded with the operands. If desired, the Am29116 can be directly connected to the S516 for loading operands to the S516. However on transferring processed data to the Am29116, a buffer is needed due to bus loading conditions and the S516 timing. Also the clock-to-output delay for the LS546 is comparable to Schottky gate delays. The actual timing diagram details can be derived from the S516 data sheet, along with the instruction and data loading/offloading sequence.

The S516 can be driven from the same microinstruction PROM as the Am29116. The only constraint on the timing is to stagger the clock to the S516. This means that if the system is running at 6 MHz, the S516 clock leads the system clock by 30-55nsec (or lags it by 112-137nsec) as shown in Figure 5. This satisfies the setup and hold timing requirements for the data and instructions. Also the processed data is clocked in the LS546 with the staggered clock. The latch of the Am29116 is loaded with the processed data. This can be done by controlling the Data Latch Enable of the Am29116. The performance is improved significantly by using the S516 with the Am29116.

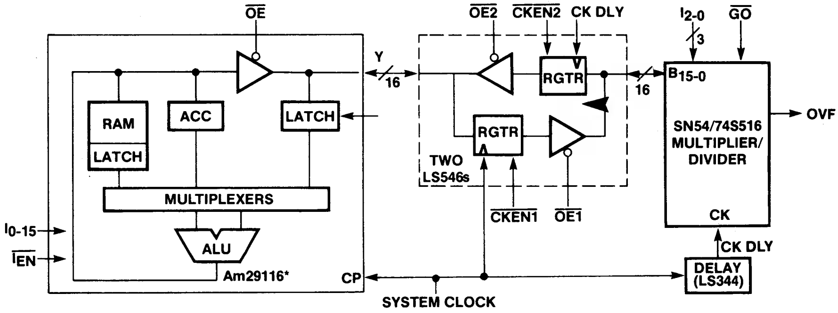
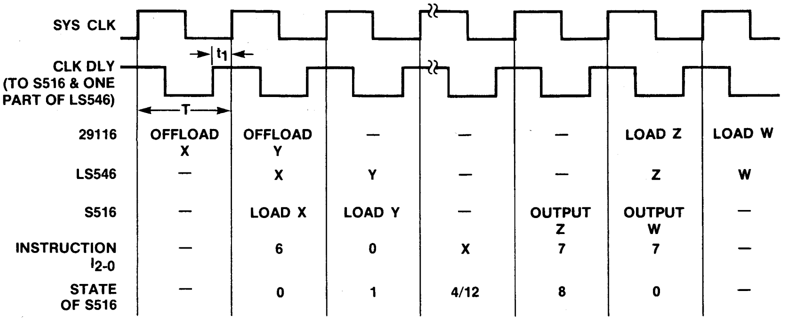


Figure 4. The Interface Diagram for the Am29116 to the S516



29116	OFFLOAD X	OFFLOAD Y	—	—	—	LOAD Z	LOAD W
LS546	—	X	Y	—	—	Z	W
S516	—	LOAD X	LOAD Y	—	OUTPUT Z	OUTPUT W	—
INSTRUCTION I_{2-0}	—	6	0	X	7	7	—
STATE OF S516	—	0	1	4/12	8	0	—

Figure 5. The Interface Timing Diagram for Multiply

The results for different operations are listed in Table 2. A point of interest is that the 29116 is only used for four clock cycles in which data is offloaded and the processed data is read. Even if the clock were stretched during the cycles the S516 is used, in case the system is normally run at speeds faster than 6 MHz, then also the combination of the Am29116/S516 performs better than the stand-alone Am29116 in terms of actual delay. If a flow-through multiplier which gives a 16-bit product every clock cycle were used instead of the S516 then the only advantage would be speed; however, there are disadvantages in terms of board area and cost. In terms of real estate, the 24-pin multiplier and three skinny-dip packages occupy less space than the large 64-pin 16-bit by 16-bit multiplier, and cost less too! Besides the S516 can also perform division; in the case of a dedicated 64-pin multiplier you have to add another PROM for a reciprocal lookup table in the case of division.

OPERATION	CYCLES	
	Am29116	Am29116 + S516
MULTIPLY (16x16)	80	13
DIVIDE (32/16)	80	26

Table 2. Performance Comparison of the Am29116/S516

The SN54/74LS546 shown in figure 4 is another product from Monolithic Memories Inc.

The S516/INTEL 8086 Interface

The INTEL 8086 is an advanced high-speed 16-bit microprocessor. It has an elaborate instruction set which does include multiply and divide instructions. However, there can be significant improvement in the performance for the multiply and divide operations, which are applicable in digital signal processing. For example in building a digital filter, one has a need to perform sum-of-products type of operation. By using a PAL16R4A to interface the S516 with an Intel 8086, one can achieve 3.4:1 improvement in performance. These calculations are based on using a 5 MHz 8086, which allows the system clock to drive the S516, after it is modified by the PAL.

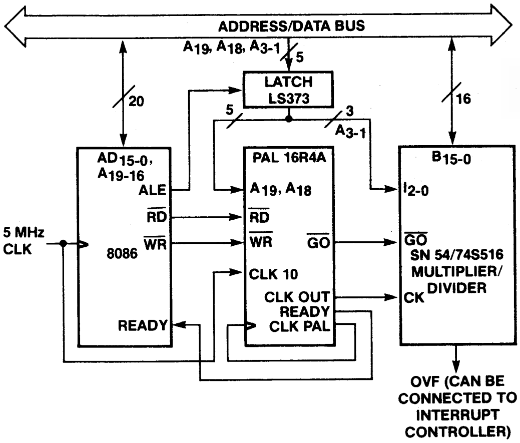


Figure 6. The Interface Diagram for the Intel 8086 to the S516

The interface circuit is shown in Figure 6. The three instruction lines to the S516 are the address lines A3,A2,A1. The most significant address bits A19, A18 are used to select the S516. The PAL is used to decode the select signal and to provide appropriate timings for $\overline{GO}$ and CLK to the S516. A simple MOVE instruction can be transformed into any of the multiply/divide routine of the S516.

Programming is simplified by assigning mnemonics to the absolute adresses for the S516 instructions:

```
INSTR0 EQU FFFF0H
INSTR1 EQU FFFF2H
INSTR2 EQU FFFF4H
INSTR3 EQU FFFF6H
INSTR4 EQU FFFF8H
INSTR5 EQU FFFFAH
INSTR6 EQU FFFFCH
INSTR7 EQU FFFFEH
```

This allows the 8086 to offload the operands by move register to memory instructions, and the results can be read from the S516 by a move memory to register instruction. In the case of multiply, there are no wait states involved, as the multiply cycles of the S516 are embedded in the 8086 instruction cycles.

The hardware interface details and programming examples are discussed in (2). Reference (2) also has a detailed PAL specification and the timing diagrams. The improvement in performance is listed in table 3.

OPERATION	CYCLES	
	8086	8086 + S516
MULTIPLY (16x16)	154	53
$\sum_{i=1}^{n} a_i x_i$ (n=4)	776	212
DIVIDE (16/16)	184	81

Table 3. Performance Comparison of the 8086/S516

Conclusion

The SN54/74S516 can significantly improve the arithmetic throughput of bipolar and MOS microprocessors. Applications to the Am29116 and 8086 have been discussed. A detailed design for enhancing the arithmetic capability of a 68000, using a S516 is discussed in (3). Square root operations can be performed to give a larger precision result, starting from a smaller precision number stored in a PROM and using the S516, referred to in (4). Summarizing, the S516 opens up microprocessor applications in many areas that demand an increased arithmetic throughput at a low overhead in real estate and in costs.

Acknowledgements

Jerry Greiner and Frank Lee, both of Monolithic Memories Inc., designed the PAL interface connecting the S516 to the 8086.

I am thankful to Ken Turkowski, of Cadlinc Inc., Palo Alto, California for his opinions on number-crunching requirements for high-performance computer graphics.

References

1. J. D. Foley and A. Van Dam, Fundamentals of Interactive Computer Graphics, © 1982 Addison-Wesley Publishing Co.

2. Jerry Greiner, Frank Lee, and Suneel Rajpal, "Enhancing 8086 Arithmetic Capability Using the 'S516 Multiplier/Divider," Monolithic Memories Application Note AN-121.

3. Richard Wm. Blasco, Vincent Coli, Chuck Hastings, and Suneel Rajpal, "SN54/74S516 Co-Processor Supercharges 68000 Arithmetic," Monolithic Memories Application Note AN-114.

4. Vince Coli, "Using a PAL to Emulate the Internal State Counter of the Monolithic Memories 'S516 LSI Multiplier/Divider," Conference Proceedings of the Eighth West Coast Computer Faire, March '83. Available from Computer Faire, 333, Swett Road, Woodside, CA 94062.

Fast 64x64 Multiplication using 16x16 Flow-Through Multiplier and Wallace Trees*

Marvin Fox, Chuck Hastings and Suneel Rajpal

Abstract

The Monolithic Memories SN54/74S556 is a high-speed fully-parallel 16x16 multiplier and it provides the entire 32-bit product on a flowthrough basis from a single part. It is available in an 84-pin Leadless Chip Carrier (LCC) and 88-pin, pin-grid array packages. 8x8 40-pin array-multipliers such as the SN54/74S557/8 have been available for several years, however there is a large parts count for implementing longer wordlengths.

This paper describes the design philosophy and internal architecture of the 'S556 and applications for larger wordlength multiplications such as 32, 48, and 64 bits using these multipliers and high-speed PROMs and ALUs also available from Monolithic Memories.

The system advantages for using the 'S556 over the MPY-16H-class multipliers is also discussed; the main advantages being the availability of the entire product each cycle and the space savings on the board.

* This paper is a slightly updated version of the paper by the same name which appeared in the Northcon/83 Professional Program Session Record, Session 24 reprint, paper 24/2, 10-12 May 1983.

Summary

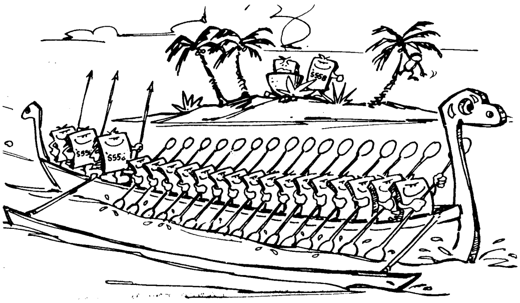
Multiplication is one basic digital-computer operation which can readily be speeded up by employing massive parallelism. "Cray multiplication" techniques, first used in large special-purpose computers a quarter of a century ago, are now commonplace in high-performance systems.

Essentially, in Cray Multiplication a full adder is placed in every position which would be occupied by a partial-product bit in a pencil-and-paper binary multiplication example (r1, r2). This technique may be applied within an LSI integrated circuit, in a system, or in both at once; it may or may not be modified by using "Booth-multiplication" approaches (r3, r4, r5).

8x8 40-pin Cray-multiplier integrated circuits have been available for several years, with a useful "flow-through" architecture. However the parts count for implementing full-blown Cray multiplication with practical scientific-computation word-lengths has been quite large. There have, of course, been several 16x16 Cray-multiplier 64-pin integrated circuits available; however, these have been unable, because of pin limitations, to furnish an entire 32-bit product in parallel. As a result, long-word-length multiplication cannot be performed economically on a flowthrough basis using these parts; some sort of clocking and multiplexing scheme is necessary to use them whenever the wordlength exceeds 16 bits, or else they must be duplicated outright.

Now there is a 16x16 Cray-multiplier part, the Monolithic Memories SN54/74S556, which provides the entire 32-bit product on a flow-through basis from a single part. The 'S556 has been designed to use the new 84-pin leadless-chip-carrier (LCC) and 88-pin pin-grid array packages, rather than compromising the architecture of the part because of the pin limitations (64 at most) of dual-in-line (DIP) packages.

This paper describes the design philosophy and internal architecture of the 'S556. It also shows how long-word-length multipliers may be built up from arrays of individual Cray-multiplier integrated circuits and programmable read-only memories (PROMs); the latter are used as "Wallace-tree" adders. Part-count and performance comparisons are made, for the representative word length of 64 bits, between implementations based on 64-pin 16x16 devices and implementations using 'S556s, in two different architectures; one which aims at lower cost and is a compromise between Cray multiplication and traditional shift-and-add multiplication.



"...MULTIPLICATION... CAN READILY BE SPEEDED UP BY EMPLOYING MASSIVE PARALLELISM"

'S556 Architecture

The 'S556, shown in Figure 1, is a 16x16 Cray multiplier designed with an ultra-high-speed array of 256 adders, internally organized to the shift-and-add technique for multiplication (r1, r2). In place of the usual ripple-carry adders used in multiplier designs to sum up the final product bits, the 'S556 uses a carry-lookahead adder.

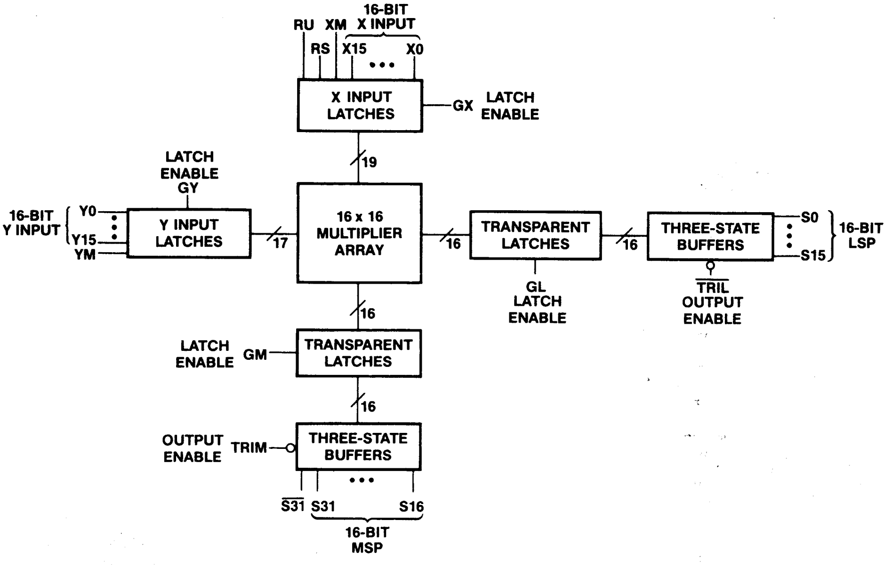


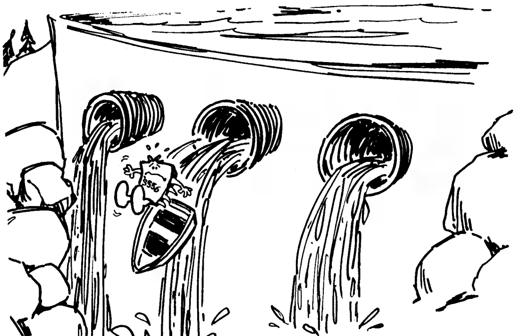
Figure 1. The 'S556 Architecture

The "flow-through" architecture of the 'S556 works equally well in synchronous or asynchronous pipelined systems. Latches are available to hold the input operands and the resulting double-length product, to increase the throughput rate in pipelined systems. If the designer does not wish to use these latches, they may be disabled, and the 'S556 then operates as a pure memoryless arithmetic network.

The 'S556 accepts operands in either unsigned or signed twos-complement form. When used in pipelined architectures, the 'S556 is capable of supplying 32-bit products at a 12.5 MHz repetitive throughput rate. The 'S556 has three-state outputs, controlled by the TRIL and TRIM control inputs.

Rounding-control input pins are provided on the 'S556 for rounding either unsigned or signed operands. Rounding is allowed in either of two binary positions, to support either "fractional-arithmetic" or "integer-arithmetic" positioning of a single-length rounded result.

The more traditional shift-and-add technique was chosen for the internal design of the 'S556 adder network because of the compactness, simplicity, and lower power requirement of this implementation. The Booth-algorithm approach, which groups the multiplier bits to effectively reduce the number of rows in the array, was considered (r3, r4, r5). However this approach also has penalties, in that it increases the width of each row from 16 to 18 bits, and the width of the final adder from 18 to 24 bits. Intrinsically, both the shift-and-add technique and the Booth-algorithm technique require 31 logic delays in the multiplier array using a ripple-carry final adder. At this point, the use of a carry-lookahead adder structure results in major speed improvements.



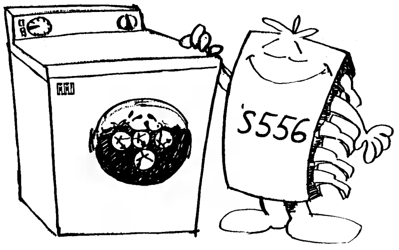
"...THE 'FLOW-THROUGH' ARCHITECTURE OF THE 'S556 WORKS EQUALLY WELL IN SYNCHRONOUS OR ASYNCHRONOUS PIPELINED SYSTEMS... "

Here again there are tradeoffs. In MSI bipolar circuits, carry-lookahead parts are reasonable to construct with scanning widths of up to 4 bits, with a carry-out available (r5). Beyond that, the circuit gets bulky and power-hungry. Parallel "banking" of 4-bit-adder groups may be used to extend this limit, but here again 4 to 5 banks is as far as this approach can be reasonably pushed. With parallel banking the 24-bit adder required by the Booth-algorithm technique can be implemented using 6 banks of 4-bit adders; this exceeds the limit of 4 to 5 banks. This shows that the Booth-algorithm implementation requires fewer horizontal rows of adders, which translates to shorter propagation delays as compared to shift-and-add technique; however the final adder in the Booth-algorithm implementation is slower than the final adder in the shift-and-add technique implementation.

The 'S556 internal design uses an Emitter-Coupled-Logic (ECL) circuit implementation, based on Monolithic Memories' new washed-emitter process. ECL was chosen here over TTL and Emitter-Follower Logic, both of which have been used in previous Monolithic Memories Cray-multiplier designs (r3, r4). Here, ECL also turns out to have the most compact circuit-layout form, requiring 82 square mils of chip surface area per full adder. Emitter Function Logic (EFL) was chosen for one portion of the design, the carry-lookahead tree, because it interfaces easily with single-ended ECL outputs. All latches are implemented in ECL, to interface easily with the TTL/ECL buffers at the inputs and the ECL/TTL buffers at the outputs. The input latches introduce one ECL delay, but there is zero additional delay at the outputs as the output latches are incorporated right into the ECL/TTL translators.

The 'S556 is a universal multiplier aimed at a flow-through-type-processor architecture. Latches are used since registers cannot implement a flow-through architecture directly.

To be sure, the currently-available 16x16 multipliers from TRW and AMD, which use 64-pin dual-in-line packages do have a feed-through capability on the output registers. This capability allows latch-like transparency on the output registers, but nowhere else, since the parts are pin-limited and input and output data must in some cases share the same pins. Such an implementation consumes considerably more chip area and power than a purely latch design.



"... THE 'S556 INTERNAL DESIGN USES MONOLITHIC MEMORIES' NEW WASHED-EMITTER PROCESS... "

Many users who wish to use registers to achieve pipelined operation can find ways to do so using the 'S556s' internal latches. Usually pipelining can be achieved by choosing the proper phasing and pulse width of the latch gate-control signals without resorting to using external registers. Of course, external registers may be used when absolutely necessary.

The 'S556 will be supplied in an 84-pin Leadless Chip Carrier (LCC), and also in an 88-pin pin-grid-array package, with an integral heat sink. Both Commercial and Military grade parts will be available. The pinout is shown in Figure 2a. A photograph of the 84-pin LCC package is shown in Figure 2b.

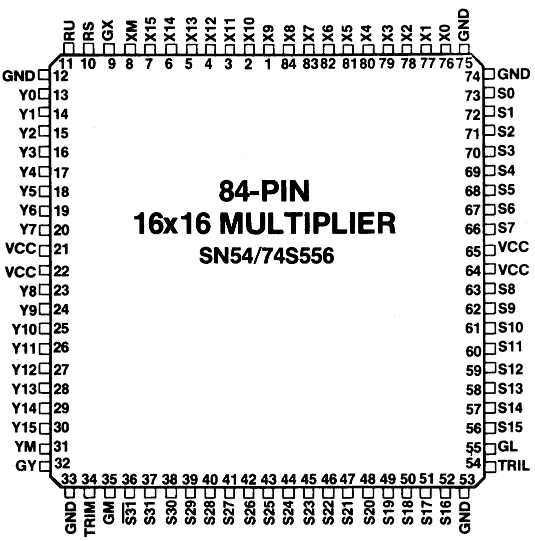


Figure 2a. The 'S556 Pinout Diagram

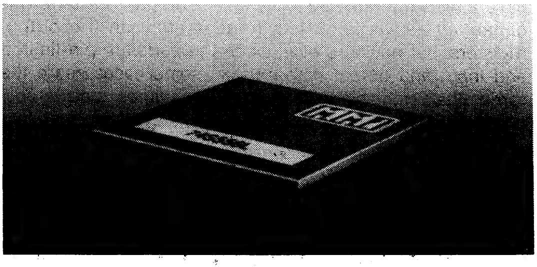


Figure 2b. The 'S556 84-pin LCC package

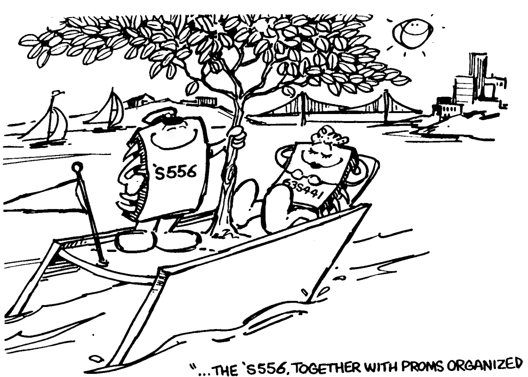
Expansion for Longer Wordlengths

A major advantage of the 'S556 is the availability of all 32 product bits in 100 nsec from the very beginning of a multiply operation, or every 80 nsec on a repetitive pipelined basis. (These times, and others quoted in this paper, are worst-case rather than typical.) Thus, the 'S556 is especially suited for longer-wordlength arithmetic units.

Other commercially-available multipliers, of the TRW MPY-16H class, are packaged in 64-pin 900-mil DIPs, which require a circuit board area of approximately 1" x 3.25." Moreover, these parts operate more slowly in expanded configurations, as the most-significant half and the least-significant half of the 32-bit double-length product must be obtained on two successive clock cycles.

Totally-Parallel 32-bit Multiplier

The 'S556, together with PROMs organized in a "Wallace-Tree" configuration, can sail along at the rate of four 56x56 multiplications every microsecond. An unsigned 32-bit multiplication can be performed using 4 'S556 multipliers, 11 63S481A PROMs used as "Wallace-Tree adders" (r1), and 16 'S381 and 5 'S182 used to form a 64-bit adder. The multipliers supply the partial products which are positioned as shown in Figure 3. The difference is that only unsigned operands are used, and only positive partial products are added. The three rows of partial products which overlap are added by using PROMs which "compress" these three rows to 2 rows, which are then added in the 64-bit adder. The compression technique is discussed in greater detail in the description, later on, of the 64-bit multiply operation. Using the above configuration, an unsigned 32x32 multiply operation can be performed in less than 200 nsec worst-case allowing for a 100-nsec 'S556 multiplier delay, a 30-nsec 63S481A PROM delay, and a 64-nsec 64-bit adder delay.



"...THE 'S556, TOGETHER WITH PROMS ORGANIZED IN A "WALLACE-TREE" CONFIGURATION, CAN SAIL RIGHT ALONG AT THE RATE OF FOUR 56x56 MULTIPLICATIONS EVERY MICROSECOND..."

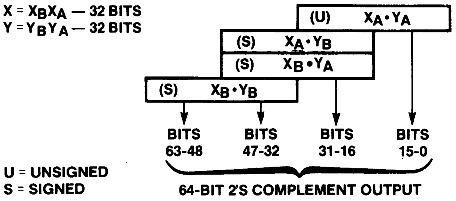
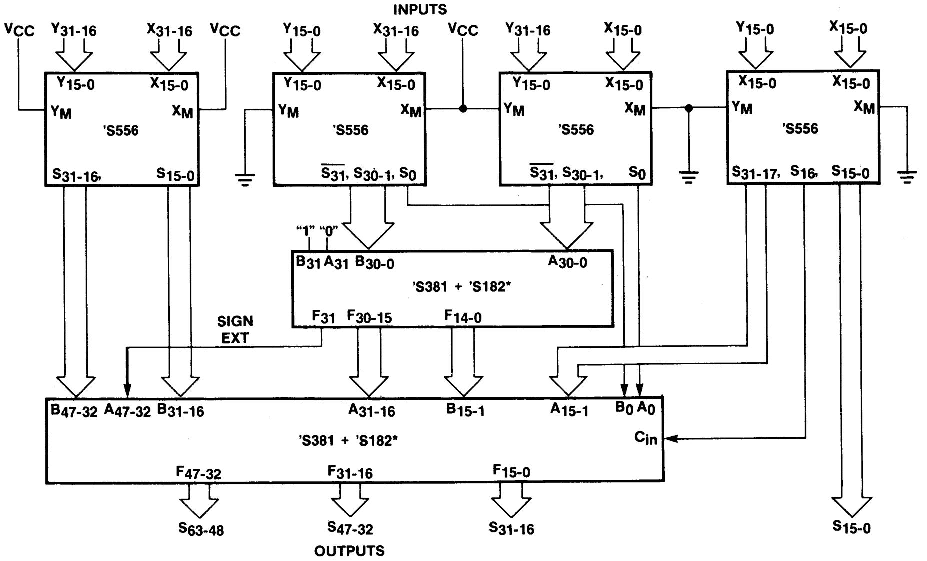


Figure 3. Partial Products for a 32x32 Multiplication

Alternatively, a twos-complement 32x32 multiplication can be performed within 228 nsec using 4 'S556s, 18 'S381s, and 7 'S182s. This 32x32 multiply operation involves the adding up of four partial products as shown in Figure 3. These four partial products are generated in four multipliers; the outputs are XA*YA, XA*YB, XB*YA, XB*YB, where X31-16 = XB, X15-0 = XA, Y31-16 = XB, Y15-0 = XA.

The implementation of this twos-complement 32x32 multiplier is shown in Figure 4. The outputs of the 16x16 multipliers are connected to two levels of adders to give a 64-bit product. The first level of adders is needed to add the two central partial products of Figure 2, XA*YB and XB*YA. Notice the technique which is used to generate the "sign extension" or the most-significant sum bit of the first level of adders. The 'S556 provides as a direct output the complement of the most-significant product bit; having this signal immediately speeds up the sign-extension computation, and reduces the external parts count.



* THESE ARE ADDER BLOCKS USING THE'S381, A 4-BIT ALU FUNCTION GENERATOR, TO PERFORM A HIGH SPEED ADD OPERATION. THE'S182 IA A LOOK-AHEAD CARRY GENERATOR AND IT REDUCES THE PROPAGATION DELAY. ALL THE ABOVE PARTS ARE AVAILABLE FROM MONOLITHIC MEMORIES INCORPORATED.

TOTAL MULTIPLY TIME = MULTIPLIER DELAY + ADDER LEVEL 1 DELAY + ADDER LEVEL 2 DELAY = 100 + 64 + 64 = 228 nsec

Figure 4. Implementation of the 32x32 Multiplier

For example, the inputs to the adder in the most significant position are the $\overline{S31}$ outputs from the two central multipliers. The sign extension of the addition of XA*YB and XB*YA is defined as

SIGN EXT $= \overline{\bar{A}.\bar{B}. + \bar{A}.C + \bar{B}.C}$, where

A is the most-significant bit of the term XA*YB;
B is the most-significant bit of the term XB*YA; and
C is the carry-in to the most-significant bits of XA*YB and XB*YA, in the adder.

The sign extension can be computed as the negation of the carry-out term of three terms, A, B, and C. This term corresponds to the negative of the carry-out of the bit position just one place to the right of the most-significant bit position of the first level of adders. The negative of the carry-out can be generated by presenting a carry-out and a binary "one" to the most significant bit of the adder. The generated sum bit then corresponds to the negation of the carry-out of the previous stage, which is the sign extension required to be added to the 16 most-significant bits of the XB*YB partial product term.

The second level of adders, which performs a 40-bit add function, is fairly straightforward. These adders can be implemented using 'S381 four-bit ALUs and 'S182 carry-bypasses ("carry-lookahead generators") which are available from Monolithic Memories, Inc. and from other vendors.

Other configurations such as 48x48 multipliers can be designed using the same methodology. Figure 5 shows the alignment of the partial products from 9 'S556s for the 48x48 case.

Serial-Parallel Multiplier

In applications where speed can be sacrificed, it is possible to implement an alternative solution using fewer multipliers, at some penalty in speed, but still with a very significant speed gain over other methods of multiplication. Figure 6 shows a plausible method of performing a 64x64 multiply operation, in four cycles. Each cycle generates four partial products, each of which is 32 bits wide; these must be added in at the appropriately-aligned bit positions to generate an 80-bit partial product, in logic external to the multipliers. On the next cycle another 80-bit partial product is generated, and is added to the previous 80-bit partial product at the appropriate alignment offset. Figure 7 shows the 16 32-bit partial products aligned appropriately to their binary weighting, for the entire time-sequenced multiply process. The final 128-bit product can be obtained from the addition of the four 80-bit partial products on successive clock cycles.

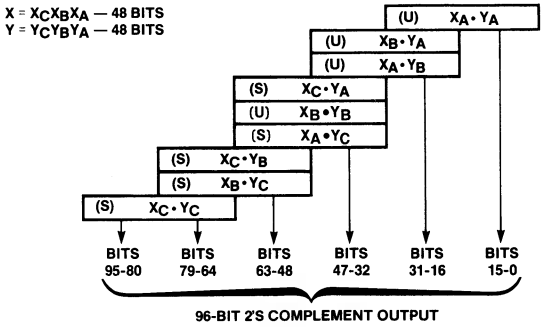
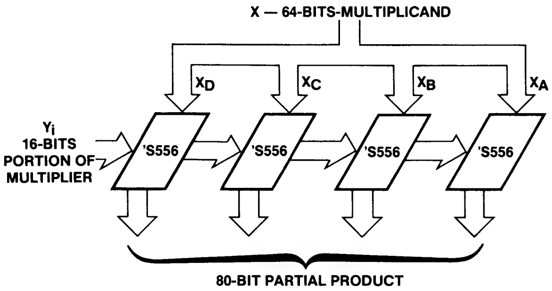


Figure 5. Partial Products for a 48x48 Multiplication



THE Y_i - PARTIAL 16-BIT OPERANDS Y_D, Y_C, Y_B, Y_A ARE LOADED AND MULTIPLIED BY THE ENTIRE 64-BIT X OPERAND IN FOUR STEPS TO OBTAIN A 128-BIT PRODUCT AS SHOWN IN FIGURE 7.

Figure 6. A Serial-Parallel Multiplier Architecture

Totally-Parallel 64-Bit Multiplier

A speed-oriented hardware configuration takes the approach of using whatever external logic is needed for the very fastest possible 64x64 multiply operation. Figure 7 may be applied in this case also; it shows 16 32-bit partial products. (For simplicity, will assume that the configuration described here deals strictly with unsigned integers, so that the 16 partial products are unsigned.) Since 16 'S556s are being used, then the 32-bit partial products corresponding to *all* of the combinations of the partitioned multiplier and the partitioned multiplicand are all available at the same time. Now comes the crucial aspect of the design, which involves adding all of these bits in at the appropriate binary positions!

Figure 8 shows the aligned configuration of the partial products for a 64x64 multiply operation. Each dot represents an output bit of the 'S556, shown at the topmost part of Figure 8. To generate the final product, these partial products must be "compressed." This compression can be achieved by grouping product bits in a logical manner, so that "deep and short" vectors are compressed to "shallower and longer" vectors. What is really being accomplished is the addition of several (here 3, 5 or 7) bits having the same weight into a simple binary sum; and then the adding up of all of these (overlapping) sums, which is normally much easier. This two-step summation is performed by a "Wallace-tree-adder" arrangement (r1, r2, r3) in which 7 vectors of varying lengths are compressed to 2 vectors, and these are in turn presented as 2 operands to a single carry-lookahead adder. The dots shown in the inverted pyramidal array in the middle of Figure 8 represent compressed outputs generated from the first level of dots. The lowermost array of dots represent the inputs to the adder; these are "compressed" outputs from the Wallace-Tree array.

For example, group A shown in Figure 8 consists of a 3x3 column of bits. If all of these bits were binary "ones" then the result when they were added would be 3 + (3.2) + (3.4) = 21, which is representable in 5 binary positions. This is precisely what "A1" signifies; a compression of a 3x3 block, A, to a 5-bit vector, A1. The compression can easily be achieved by using a PROM, with the 9 bits of A as address lines, and the outputs as A1. The PROM used in this example is the Monolithic Memories 63S481A 30-nsec 512x8 PROM; only five of the eight output bits are used. Designers may prefer to group the bits in a different configuration from the one suggested in Figure 8; many other arrangements are possible. For example, one may group another column of three bits and thereby reduce a 4x3 block to a 6-bit vector, using 63S3281s, which are (40-nsec 4Kx8 PROMs); this approach would give a different pattern than the one in the middle of Figure 8.

Similar compressions for Group B to B1 can be performed using 63S441/1A 1Kx4 PROMs. This configuration compresses five 2-bit vectors to a 4-bit vector, which fits the 10-bit input address and 4-bit output word of the 1Kx4 PROM.

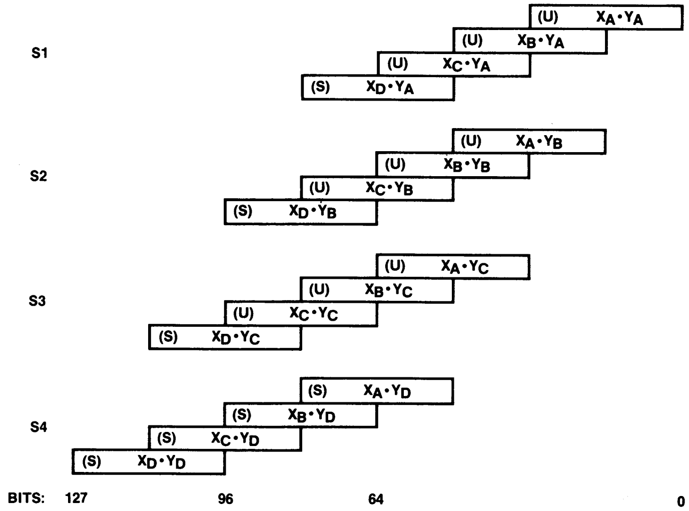


Figure 7. Partial Product Alignment for a Serial-Parallel Multiplier

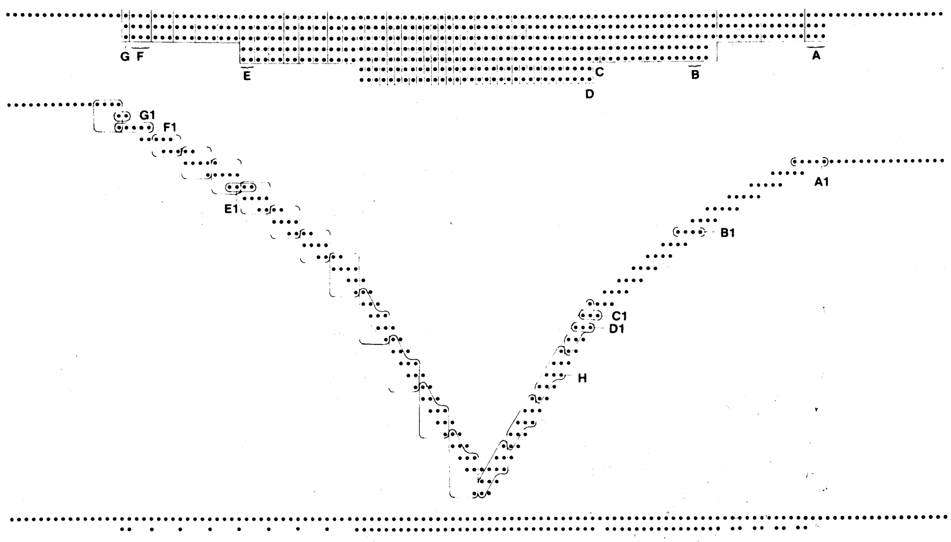


Figure 8. Partial Product/Bit Grouping for a Totally Parallel 64x64 Multiplier

Other compressions shown are C and D groups to C1 and D1 respectively. The C group is handled by compressing five 1-bit vectors to a 3-bit vector. The D group is handled by compressing seven 1-bit vectors to a 3-bit vector. The C and D groups can be compressed using 'S141/1A 256x4 PROMs. Similarly, groups E, F, G, and H are compressed to E1, F1, G1 and H1 respectively. All the above mentioned PROMs are available from Monolithic Memories.

The second level of dots has some groups of four columns. These four-column groups contain 3 bits in the least-significant bit position, and 2 bits in the remaining columns. These 9 inputs can be compressed using a 63S481A 30-nsec 512x8 PROM, to a vector 5 bits wide. For parts-counting purposes, the same 63S481A PROM type is used for all the compressions in the middle of Figure 8.

To aid users in the programming of PROMs for these and other Wallace-tree applications, or in fact any other applications exploiting PROMs as logic elements, Monolithic Memories provides *P*rogrammable *L*ogic *E*lement *AS*se*M*bler (PLEASM), a portable computer program written in FORTRAN. PLEASM provides a simple method for generating a PROM truth table. The user has only to supply equations which define the arithmetic/Boolean function needed within the PROM; PLEASM does all the drudgery of figuring out the code values which are needed in each PROM location.

Sample PLEASM source codes are shown at the end of this paper. For example, the entire 1Kx4 PROM which reduces the five 2-bit vectors to a 4-bit vector can be specified, using PLEASM, in 15 or fewer lines of code. Without PLEASM or its equivalent, the user would have had to specify the contents of 1024 PROM locations, after computing the corresponding code values for those locations.

Performance Comparisons

The bottom line for any hardware-architecture analysis is how fast the system runs, and what it costs in circuit-board real estate and dollars. With this understanding, a performance table is derived, based on three configurations.

The first is the configuration of Figure 7, using 4 'S556 multipliers; the entire multiplication takes four clock cycles. In addition to the multiplier ICs, a 64-bit adder is needed for the four partial products, which effectively furnishes a 80-bit partial product on every cycle. A 64-bit adder can be used to do the addition, since the least-significant partial-product bits are available directly. The 80-bit partial product has to be shifted 16 bits and then added to the second 80-bit partial product, which implies a need for a 64-bit register and an 80-bit ALU, which together serve as an accumulator.

The second configuration is the totally-parallel design using 16 'S556 multipliers plus PROMs and ALUs, shown in Figure 8.

The third configuration uses TRW-MPY16H-class 64-pin 16x16 multipliers. The entire 32-bit product of an MPY-16H is available on two successive clock cycles, as the product lines are shared with the incoming data. An additional 145 nsec is added to the MPY-16H time to allow for the necessary clocking and multiplexing steps to occur: effectively, the operands cannot be pipelined at one clock cycle as may be done in the 'S556 architecture. Even if the pin-compatible Am29516 multiplier is used, a cycle is still wasted, as two cycles are needed to clock the entire multiplier.

There is one way around this problem, when using the Am29516 multiplier; twice as many multipliers are used, and a pair of adjacent multipliers receive the same input operands. One multiplier of the pair then outputs the least-significant half of the product, and the other multiplier of the pair outputs the most-significant half of the product; thus, the two paired Am29516 64-pin DIPs are functionally a quasi-equivalent of the 84-pin 'S556, albeit they require many times the circuit board area.

The analysis in the Table 1 assumes the use of 16 MPY-16HJ multipliers. 16 16-bit registers are needed to hold the 16 halves of the various different partial products. After the

16 32-bit products are available, then the Figure 8 configuration is applicable to this case as well. In terms of speed, it is assumed that the MPY-16HJ multiplier configuration takes a clock cycle (145 nsec), more than the computed delay. The computed delay in this case is that of the MPY-16HJ in its feedthrough mode, followed by that of the compressor array of Figure 8.

Multiply Configuration	Speed	Components Used
64x64 Multiply Serial-Parallel	4x240 nsec	4 'S556s 36 'S381s 11 'S182s 8 'S374s
64x64 Multiply Totally-Parallel Using 'S556s (84-pin packages)	251 nsec	16 'S556s 27 63S481As (512x8) 16 63S441s (1Kx4) 33 63S141s (256x4) 32 'S381s 11 'S382s
64x64 Multiply Clocked Parallel Using MPY-16HJ (64-pin packages)	481 nsec	16 MPY-16HJ 32 'S374s 27 63S481s (512x8) 16 63S441s (1Kx4) 33 63S141s (256x4) 32 'S381s 11 'S382s

Table 1. Performance Comparisons

Conclusion

The 'S556 16x16 multiplier is an excellent building block for longer-wordlength multipliers. It is useful in graphics systems, array processors, minicomputers, and large mainframe computers. It surpasses the currently-available 64-pin-DIP multipliers in that the *entire* 32-bit product is available on every clock cycle.

Some configurations which use 'S558-type 8x8 multipliers as building blocks for a 56x56 multiplier are discussed in r1 and r2.

References

All of the following references are available from Monolithic Memories, Inc.

r1. "Big, Fast, and Simple—Algorithms Architecture, and Components for High-End Superminis," Ehud Gordon and Chuck Hastings, Monolithic Memories Application Note AN-111.

r2. "How to Design Superspeed Cray Multipliers with '558s," Chuck Hastings, Monolithic Memories Application Note, incorporated into the SN54/74S557/8 data sheet.

r3. "Real-Time Processing Gains Ground with Fast Digital Multiplier," Shlomo Waser, *Electronics*, 9/29/77.

r4. "State-of-the-Art in High Speed Arithmetic Integrated Circuits," Shlomo Waser, *Computer Design*, 6/1978.

r5. "Doing Your Own Thing in High-Speed Arithmetic," Chuck Hastings, *Conference Proceedings of the 6th West Coast Computer Faire,* pages 492-510, 4/5/81. Also Monolithic Memories Conference Proceedings reprint CP-102.

Appendix — Sample PLEASM Source Listing — To Reduce Group B In Figure 8

```
PROM1024X4                                         PROM DESIGN SPECIFICATION
PATTERN NUMBER 2                                       VINCENT COLI 08/22/82
PARTIAL PRODUCTS ADDER
MMI SUNNYVALE, CALIFORNIA
.ADD A0 A1 B0 B1 C0 C1 D0 D1 E0 E1
.DAT P0 P1 P2 P3


P3,P2,P1,P0 = A1,A0 .+. B1,B0 .+. C1,C0 .+. D1,D0 .+. E1,E0


FUNCTION TABLE

A1 A0 B1 B0 C1 C0 D1 D0 E1 E0 P3 P2 P1 P0

;AA    BB    CC    DD    EE     PPPP      COMMENTS
;10    10    10    10    10     3210      A + B + C + D + E = P
-----------------------------------------------------------
 LL    LL    LL    LL    LL     LLLL      0 + 0 + 0 + 0 + 0 =  0
 LH    LH    LH    LH    LH     LHLH      1 + 1 + 1 + 1 + 1 =  5
 HL    HL    HL    HL    HL     HLHL      2 + 2 + 2 + 2 + 2 = 10
 HH    HH    HH    HH    HH     HHHH      3 + 3 + 3 + 3 + 3 = 15
-----------------------------------------------------------


DESCRIPTION

This 1Kx4 PROM performs the partial products reduction
for a Wallace tree adder.


                                  A1 A0
                                  B1 B0
                                  C1 C0
                                  D1 D0
                               +  E1 E0
                            ------------
                            P3 P2 P1 P0
```

Northcon/83
Electronics Show & Convention
May 10-12, 1983
Portland, Oregon

Cascade Chapter, ERA
Portland and Seattle Sections IEEE
Portland and Seattle Chapters, NWPCA

The Design and Application of a High-Speed Multiply/Divide Board for the STD Bus. Northcon/82 Session 15

Michael Linse, Gary Oliver, Kirk Bailey,
Michael Alan Baxter

Abstract

A fundamental limitation in most microcomputer systems is high-speed arithmetic computing speed, especially when multiplications or divisions are required. A hardware multiply/divide board designed to work efficiently with a STD BUS microcomputer in an industrial control system is presented.

The application described includes the simultaneous calculation of several digitally-controlled servo loops which allow control of machinery to within the resolution of servo position sensors at a bandwidth that software alone cannot accomplish.

Introduction

The development of industrial controls that were able to control an entire process, and yet be inexpensive, came with the minicomputer. The relatively high processing speed that allowed multiplexing many sensors extended the reach of what could be done by an integrated process control system. The low cost (as compared to supercomputers) and modularity of the minicomputer made it widely available. Because the software for a control system could be easily changed, many upwards-compatible control systems could be built around the same minicomputer. The minicomputer today, however, is expensive and slow.

With the availability of the microcomputer, many new control tasks can be accomplished at a fraction of the cost of minicomputer based systems. The more recent generations of microcomputers now have exceeded the processing power of minicomputers. Control algorithms written for minicomputers now can run as easily on a microcomputer. But new technological advances in microcomputer components have a potentially disastrous side effect. There is a continuously rising expectation among users that a newer computer will better solve complex problems. Usually, no regard is given for the architecture of the particular computer and whether or not that architecture is suited for the desired application. Most general-purpose microcomputers have one serious deficiency—arithmetic computing speed. The rate at which a microcomputer can compute multiplications or divisions limits the bandwidth of a system for many control algorithms. There is a compromise solution to this problem. The use of a dedicated arithmetic coprocessor can significantly aid computational bandwidth.

A dedicated coprocessor operates in parallel with the microcomputer, and in addition to increasing arithmetic computation speed, a coprocessor effectively adds new instructions to the microcomputer. A coprocessor that utilizes operands and data through a memory mapped interface transforms simple 'move' instructions into powerful arithmetic instructions.

Basic Operation

The STD Multiply/Divide Board is designed around the Monolithic Memories, Inc., 74S516 LSI Multiply/Divide IC[1] The Board operates in parallel with the STD BUS[2] computer system (in this case a Z-80) and appears as a memory-mapped device on the STD BUS. A 512-byte addressing space is selected by the user anywhere within 64K memory space. Alternatively, the Board can use the STD BUS MEMEX line to select an address block within 128K addressing space.

A simplified block diagram of the Board is shown in Fig. 1. The Board consists of STD BUS data and address buffers, two memories, the mutliply/divide IC, and a microcoded control system. STD BUS address bits A15-A9 and MEMEX select a 512-byte address block for the Board; A6-A0 are used by the control system to enable loading data, start a multiply, etc. The control system also sequences the multiply/divide IC. The Data From Processor and Data To Processor RAM allow a byte/word conversion between STD BUS and the multiply/divide IC which uses 16-bit words.

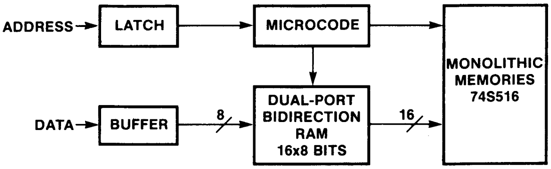

ADDRESS
LATCH
MICROCODE
DATA
BUFFER
8
DUAL-PORT BIDIRECTION RAM 16x8 BITS
16
MONOLITHIC MEMORIES 74S516

Fig. 1. STD Multiply/Divide Block Diagram.

The user implements a given mathematical operation on 16-bit data by passing the operands as two bytes sent through a defined pair of addresses. The addresses used uniquely define the function to be performed by the Board. The order of the two bytes is defined as low-byte, then high-byte; this choice reinforces the protocol demanded by the Z80 CPU, as well as high-level language software tools that have 16-bit word constructs for 8-bit microprocessors. Operations such as a 16-by-16 multiply produce 32-bit results, which are read out of the Board as four bytes, lowest to highest, in order of significance. The Board appears to the programmer essentially as a write-only memory for operands and a read-only memory for computed results. No handshake or 'done' signal is available and the board does not generate interrupts to the Z80. Interrupt processing is fairly slow when compared to the speed of the Board (this information is given later). The user would also typically not want to poll a heavily used, very fast coprocessor.

Many variants of multiplication and division operations are available on the Board, allowing the programmer the option to choose a function that fits the needs of the process. The Board uses only 16-bit integer representation for data (except in the case of double-length results); the 74S516 multiplier/divider does allow data to represented as 16-bit fractions. Due to the needs of the application of the Board the fractional arithmetic feature of the 74S516 is not used. The various operations the Board performs are listed in Table 1. The execution times are based on a micro-instruction cycle time that is the same length as the clock to the 74S516. The nomenclature in the table is compatible with the Monolithic Memories, Inc. data sheet; X and Y are 16-bit data, and Z/W 32-bit data or results.

Address*	Operation	Input Order	Cycle (μs)
08/09	X•Y	X, Y	3.84
0A/0B	(Z,W)/X	Z, W, X	8.00
0C/0D	−X•Y	X, Y	3.84
0E/0F	X•Y+Z	X, Y, Z	4.16
10/11	−X•Y+Z	X, Y, Z	4.16
12/13	X•Y+(Z, W)	X, Y, Z, W	4.48
14/15	−X•Y+(Z, W)	X, Y, Z, W	4.48
16/17	Z/X	Z, X	8.32
18/19	W/X	W, X	8.32
00/01	READ LOW	n/a	n/a
02/03	READ HIGH	n/a	n/a

*The addresses referenced in Table 1 are for the bottom 8 bits of the address only; two numbers indicate the low/high byte address used.

Table 1. Operations and cycle times.

Technology Implementation

The primary consideration in the design of this arithmetic board was speed of operation. The second most important consideration was the ease of programming. The Board had to be approximately the size of STD BUS format cards (6.5 in. by 4.5 in.). The arithmetic performed had to use 16-bit integers, and handling of 32-bit results had to be accomodated without losing speed performance.

Bipolar LSI and MSI logic was used throughout the card. Speed requirements were met by using Schottky TTL as well as Low-Power Schottky for slower moving sections of the design. Additional SSI logic was used to create timing and control signals to set up events between MSI elements in as short of a time delay as possible. Despite the wide availability of bit-slice processors and controllers, the best solution to the design appeared to be use of a MSI/SSI control structure, LSI memories for data and microinstruction storage, and the single multiply/divide IC as the arithmetic element. The large package size of bit-slice parts and the number of needed parts to perform the arithmetic would become exessive in terms of board area. The control structure needed to also be flexible to allow easy changes if needed during the debugging of the prototype. Thus, a microcode was devised, and further simplified by making all operations performed by the Board required to fit within the same number of microinstruction words. Only 16-bits were required for each micro-instruction, thus allowing all microprograms to fit within two PROM devices; a few of the bits were spares for expansion or correction of errors.

Arithmetic performance of the card (summarized in Table 1) at the time of its design was dependent on two speed-critical elements: the 74S516 and the microinstruction PROM access time. The 74S516 (then designated as '67516') did not have 100 ns clock cycle times for all operations, and large (512 x 8) PROMs were relatively new and had 80 ns access times. The microcycle of the Board has been set for 320 ns in current implementation; the design is sufficiently fast to allow for 100 ns microcycles; the new PROMs and current versions of the 74S516 allow this speed, and other than these new parts, only a single component change is required.

Four-layer circuit boards were used and consistent and regularized component placement resulted in the ability to put 51 ICs on a slightly extended STD BUS format circuit card. Since input/output type cards are typically longer than the 6.5 in. dimension to allow room for connector access, the 7.7 in. card turned out to be 0.2 in. longer than other cards in the system, which resulted in no difficulties. 15 mil conductor line-widths and a minimum of 10 mil spacing between conductors allowed a 10 MHZ board to be manufactureable by a number of PC shops. Inner layer power supply distribution for +5 VDC and Ground resulted in extremely low current spiking noise, a necessity for reliable system operation on a card requiring 2.5A or more in less than 34 square in. of board space. The 74S516 required heat-sinking, and the installation of the Board is such that a 250 CFM fan blows air from underneath the card; no heat dissipation problems arose from any of the components.

Application

The Multiply/Divide Board is used in a Z80 STD BUS computer system to perform high-speed calculation of several Proportional-Integral-Differential (PID) servo control loops. The servo loops sense hydraulic positioners that can move 1000 lb. masses at greater than 10 in. per second. The relability and loop stability of such a system cannot be understated. The software that is used in this application requires efficiency and high-speed arithmetic. The Board allowed the software to essentially use 16 and 32-bit arithmetic, albeit bus transfers are 8-bits wide; the 16-bit constructs in high-level language allow time-critical code to use the Board with little overhead. The response time for each loop is under 2 msec and only 1 Board is required in the system to run several loops concurrently.

References

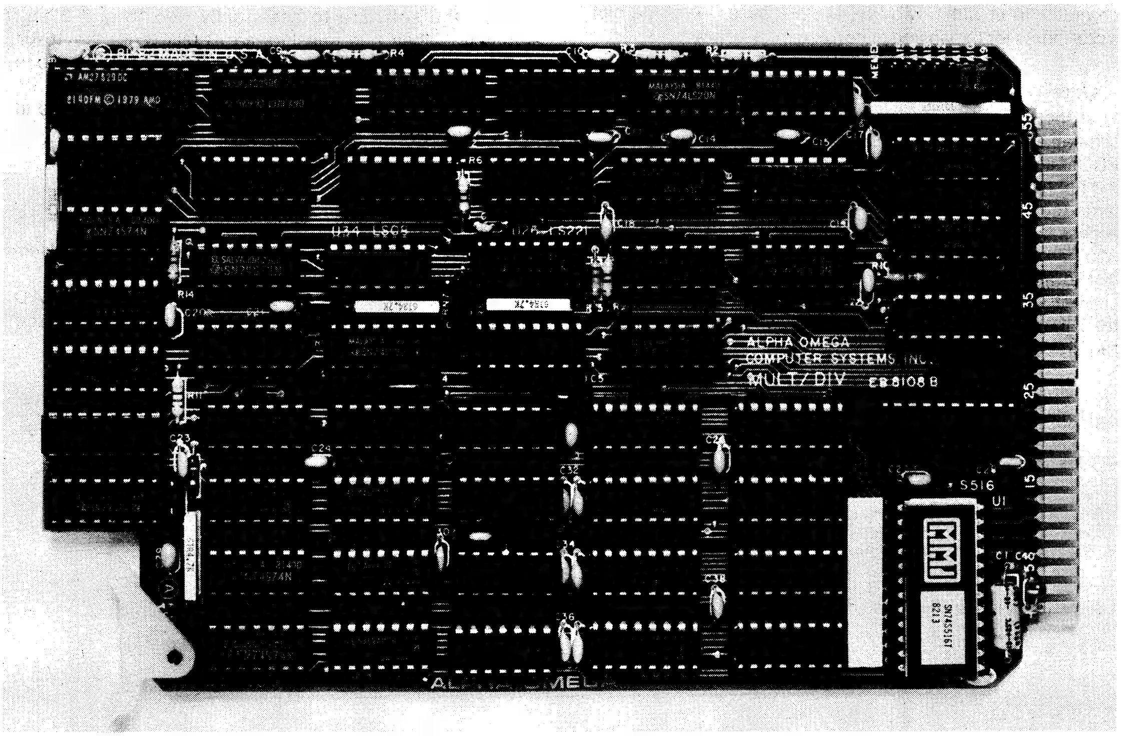
1. __________, *16 x 16 Multiplier/Divider SN54/74S516, Monolithic Memories, Inc.*, August 1981 Data Sheet.
2. __________, *STD BUS Specification and Practice,* STD Manufacturers Group (STDMG), Jan. 1981, Pro-Log Corporation.

This paper courtesy of:

Michael Linse, Senior Design Engineer
Gary Oliver, Senior Software Design Engineer
Kirk Bailey, System Analyst
Michael Alan Baxter, Design Engineer
Alpha Omega Computer Systems, Inc.
P.O. Box U
Corvallis, OR 97339

Northcon/82

Northwest Electronics Show and Convention
Seattle Center Coliseum/May 18-20, 1982



Shaft Encoder

Willy Voldan

The trend away from analog systems to digital numerical control systems focuses new attention on instruments that convert the analog output of position sensing devices into digital format.

The recording of the position of moveable parts of a machine requires high accuracy, as well as noise immunity, which is possible to attain through shaft encoder circuits of the type used in speed controllers and optical position sensing devices.

This application covers the principles of shaft encoding and a possible circuit solution, using PAL for minimum chip count.

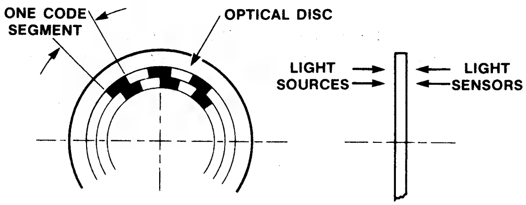
USING PAL® to GET RELIABLE DIGITAL DATA FROM SHAFT ENCODER CIRCUITS

The trend away from analog systems to digital numerical control systems focuses new attention on instruments that convert the analog output of position sensing devices into digital format.

The recording of the position of moveable parts of a machine requires high accuracy, as well as noise immunity, which is possible to attain through shaft encoder circuits of the type used in speed controllers and optical position sensing devices.

Principles of Shaft Encoding

Most common used are opto electrical encoders. The advantages of this type of circuits are direct shaft to digital encoding, a minimum number of power supplies, low power requirements, low cost and high speed.



Optical encoders measure shaft rotation by detecting the light which passes through a rotating code disc and a fixed slit.

It consists of a number of light sources and sensors whose paths are interrupted by a disc that has transparent and apaque areas.

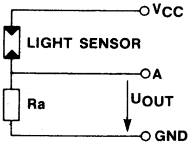
By using various concentric patterns, the shaft position can be defined. Light shines through the disc onto a sensor. The sensor turns on when the light passes through a transparent part, giving an output depending on the opacity of the segment. The combined output of all the segments is a digital information representing the disc and the shaft position.

— an **ABSOLUTE ENCODER** has a number of concentric tracks on the code disc, which provide a whole word parallel readout of shaft angle without the necessity of counting pulses. The code patterns on the disc are a kind of storage.

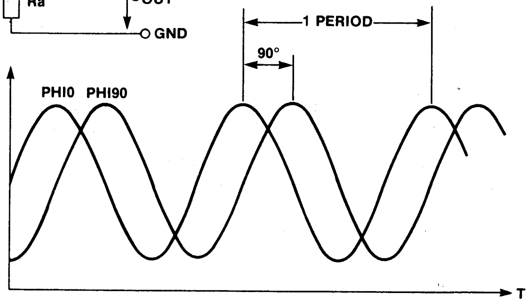
Therefore the readout is also present after a power interruption.

— **INCREMENTAL ENCODER**s, the type which will be described here, have a single code track on the disc, and angular position is determined by counting pulses produced by the modulated light falling onto the photodetectors.

Direction sensing is obtained by the use of quadrature signals which are provided by appropriate phasing of the paths.

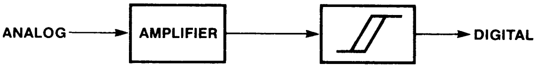


CIRCUIT WHICH PRODUCES MODULATED OUTPUT SIGNALS USING LIGHT SENSORS (PHOTO DIODES)



Therefore, if the power is interrupted an INCREMENTAL ENCODER must have the zero position reestablished.

The two signals gained from the light sensors have to be amplified and converted into digital format using Schmitt-Triggers.

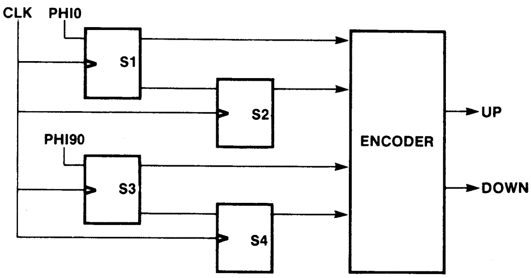


The two signals are dephased from each other by a quarter of a period. The direction of the movement can be determined if one signal leads or lags the other.

Out of his phase relationship the shaft encoder produced the information about direction and speed for the following UP/DOWN-counter.

To avoid random discrepancies during switching operations it is recommended to build shaft encoders using synchronous clocked circuitry.

The picture below shows a typical circuit diagram for SHAFT ENCODING:



The phasing of the two signals PHI0 and PHI90 is discriminated by the four registers and their outputs are encoded.

The principle of the above circuit is the time delay between the clock pulses for the two signals PHI0 and PHI90.

S1 = PHI0 clk S2 = PHI0 clk + 1

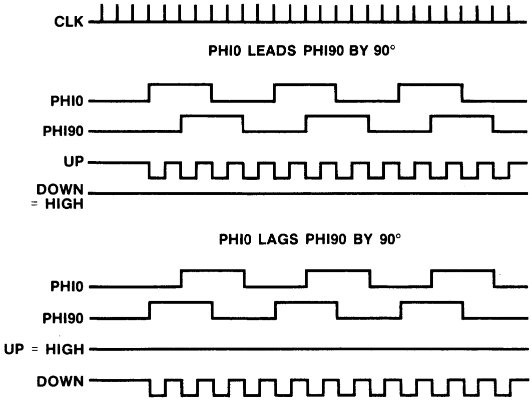
S3 = PHI90 clk S4 = PHI90 clk + 1

A transition in the signals PHI0 and PHI90 from L to H or vice versa causes a change in the data stored in the registers which is then encoded for direction sensing.

The logic equations for the encoder circuitry are given below

PHI0 leads PHI90:	S1	*	S2	*	S3	*	/S4
	/S1	*	/S2	*	/S3	*	S4
	S1	*	/S2	*	/S3	*	/S4
	/S1	*	S2	*	S3	*	S4
PHI0 lags PHI90:	/S1	*	/S2	*	S3	*	/S4
	S1	*	S2	*	/S3	*	S4
	S1	*	/S2	*	S3	*	S4
	/S1	*	S2	*	/S3	*	/S4

* stands for AND-Function.



Let us assume that when the shaft rotates in a clockwise direction, the input PHI0 leads the input PHI90 by 90° and the logic will generate pulses only at the UP-output.

On the other hand, when the shaft's rotation is in the counterclockwise direction the input PHI0 lags the input PHI90 by 90°.

In this case, four pulses per square wave are presented at the DOWN-output. Note that both the UP and DOWN outputs of the counter are normally held high.

To ensure that no shaft encoder transition is missed, the clock frequency should be at least 8 × n × s, where n is the number of pulses produced by the encoder for each shaft revolution and s is the maximum speed in revolution per second to be expected.

Most common used are CLK frequencies in the range of 1MHz for optimum circuit operation.

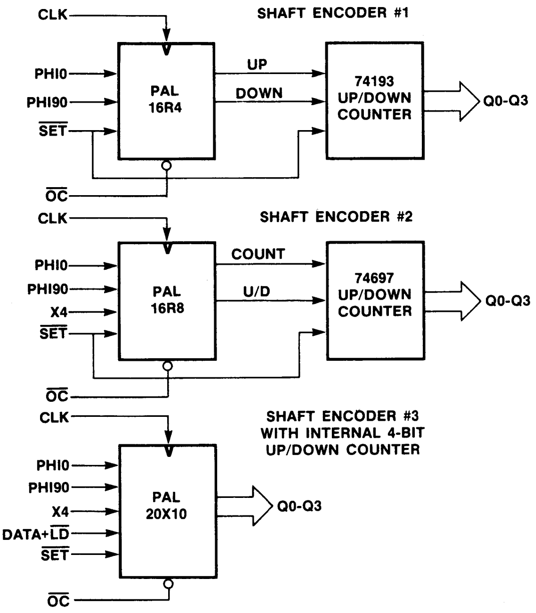
Synchronous two channel shaft encoders as described above are relatively insensitive to encoder phase errors. They use no temperamental Mono-Flops and can detect the occurence of illegal transition states generated by the circuit.

It is the property of such kind of circuits that interference on the input lines are ignored if they occur between two clock cycles.

Random noise on both input lines results in UP and DOWN count so that in the end the digital information remains unchanged.

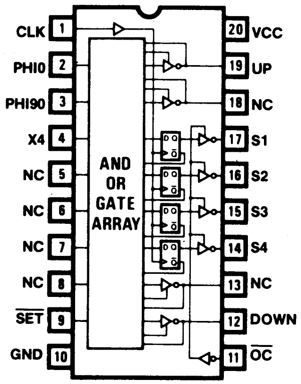
Hence synchronous SHAFT ENCODERS are extremely useful in electrically noisy environments.

The following PAL Applications are examples of such circuits using a single PAL.



```
PAL16R4                                                 PAL DESIGN SPECIFICATION
P7016                                                     WILLY VOLDAN 09/09/82
SHAFT ENCODER No. 1
MMI GMBH MUNICH
CLK PHI0 PHI90 X4 NC NC NC NC /SET GND
/OC DOWN  NC   S4 S3 S2 S1 NC  UP  VCC


/S1 :=  /PHI0                                                ;CHECK FOR PHI0
     +   SET                                                 ;INITIALIZE S1=L

/S2 :=  /S1                                                  ;CHECK FOR S1
     +   SET                                                 ;INITIALIZE S2=L

/S3 :=  /PHI90                                               ;CHECK FOR PHI90
     +   SET                                                 ;INITIALIZE S3=L

/S4 :=  /S3                                                  ;CHECK FOR S3
     +   SET                                                 ;INITIALIZE S4=L

IF (VCC) /DOWN =  S1* S2* S3*/S4* PHI0* PHI90                ;PHI0 LEADS PHI90
               + /S1*/S2*/S3* S4*/PHI0*/PHI90                ;PHI0 LEADS PHI90
               +  S1*/S2*/S3*/S4* PHI0*/PHI90                ;PHI0 LEADS PHI90
               + /S1* S2* S3* S4*/PHI0* PHI90                ;PHI0 LEADS PHI90

IF (VCC) /UP   = /S1*/S2* S3*/S4*/PHI0* PHI90                ;PHI90 LEADS PHI0
               +  S1* S2*/S3* S4* PHI0*/PHI90                ;PHI90 LEADS PHI0
               +  S1*/S2* S3* S4* PHI0* PHI90                ;PHI90 LEADS PHI0
               + /S1* S2*/S3*/S4*/PHI0*/PHI90                ;PHI90 LEADS PHI0
```

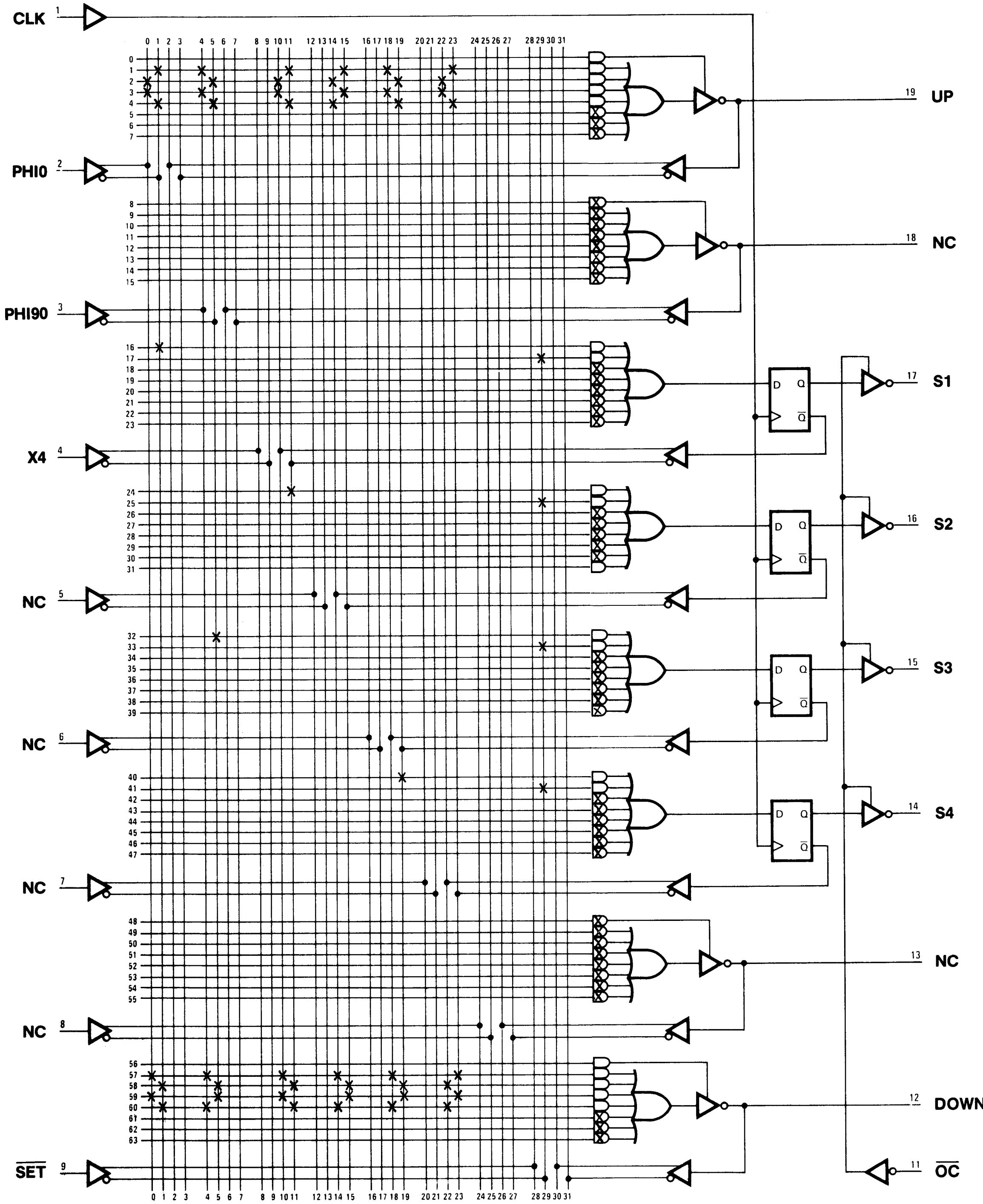
PAL16R4



```
FUNCTION TABLE
CLK /OC /SET PHI0 PHI90 S4 S3 S2 S1 UP DOWN
;---CONTROLS---         --INPUTS--     SSSS     OUTPUTS
;CLK  /OC  /SET         PHI0 PHI90     4321     UP DOWN     COMMENTS
-----------------------------------------------------------------------------
  C    L    L            X     X       LLLL     H    H      CLEAR REGISTERS
  C    L    H            L     L       LLLL     H    H
  C    L    H            L     L       LLLL     H    H
  C    L    H            L     H       LHLL     L    H      COUNT UP
  C    L    H            L     H       HHLL     H    H
  C    L    H            H     H       HHLH     L    H      COUNT UP
  C    L    H            H     H       HHHH     H    H
  C    L    H            H     L       HLHH     L    H      COUNT UP
  C    L    H            H     L       LLHH     H    H
  C    L    H            L     L       LLHL     L    H      COUNT UP
  C    L    H            L     L       LLLL     H    H
  C    L    H            L     H       LHLL     L    H      COUNT UP
  C    L    H            L     H       HHLL     H    H
  C    L    H            H     H       HHLH     L    H      COUNT UP
  C    L    H            H     H       HHHH     H    H
  C    L    H            H     L       HLHH     L    H      COUNT UP
  C    L    H            H     L       LLHH     H    H
  C    L    H            L     L       LLHL     L    H      COUNT UP
  C    L    H            L     L       LLLL     H    H
  C    L    H            L     H       LHLL     L    H      COUNT UP
  C    L    H            L     H       HHLL     H    H
  C    L    H            H     H       HHLH     L    H      COUNT UP
  C    L    H            H     H       HHHH     H    H
  C    L    H            H     L       HLHH     L    H      COUNT UP
  C    L    H            H     L       LLHH     H    H
  C    L    H            L     L       LLHL     L    H      COUNT UP
  C    L    H            L     L       LLLL     H    H
  C    L    L            X     X       LLLL     H    H      CLEAR REGISTERS
  C    L    H            L     L       LLLL     H    H
  C    L    H            L     L       LLLL     H    H
  C    L    H            H     L       LLLH     H    L      COUNT DOWN
  C    L    H            H     L       LLHH     H    H
  C    L    H            H     H       LHHH     H    L      COUNT DOWN
  C    L    H            H     H       HHHH     H    H
  C    L    H            L     H       HHHL     H    L      COUNT DOWN
  C    L    H            L     H       HHLL     H    H
  C    L    H            L     L       HLLL     H    L      COUNT DOWN
  C    L    H            L     L       LLLL     H    H
  C    L    H            H     L       LLLH     H    L      COUNT DOWN
  C    L    H            H     L       LLHH     H    H
  C    L    H            H     H       LHHH     H    L      COUNT DOWN
  C    L    H            H     H       HHHH     H    H
  C    L    H            L     H       HHHL     H    L      COUNT DOWN
  C    L    H            L     H       HHLL     H    H
  C    L    H            L     L       HLLL     H    L      COUNT DOWN
  C    L    H            L     L       LLLL     H    H
  C    L    H            H     L       LLLH     H    L      COUNT DOWN
  C    L    H            H     L       LLHH     H    H
  C    L    H            H     H       LHHH     H    L      COUNT DOWN
  C    L    H            H     H       HHHH     H    H
  C    L    H            L     H       HHHL     H    L      COUNT DOWN
  C    L    H            L     H       HHLL     H    H
  C    L    H            L     L       HLLL     H    L      COUNT DOWN
  C    L    H            L     L       LLLL     H    H
  X    H    X            X     X       ZZZZ     X    X      TEST HI-Z
```

Shaft Encoder No. 1

Logic Diagram PAL16R4



```
PAL16R8                                                 PAL DESIGN SPECIFICATION
P7095                                                      WILLY VOLDAN 09/09/82
SHAFT ENCODER No. 2
MMI GMBH MUNICH
CLK PHI0 PHI90 X4 NC NC NC NC /SET GND
/OC UD NC S4 S3 S2 S1 NC COUNT VCC


/S1     := /PHI0                                      ;CHECK FOR PHI0
         +  SET                                       ;INITIALIZE S1=L

/S3     := /PHI90                                     ;CHECK FOR PHI90
         +  SET                                       ;INITIALIZE S3=L

/S2     :=  S1                                        ;CHECK FOR /S1
         +  SET                                       ;INITIALIZE S2=L

/S4     :=  S3                                        ;CHECK FOR /S3
         +  SET                                       ;INITIALIZE S4=L

/COUNT :=   S1* S2*/S3* S4                            ;THIS OUTPUT ALTERNATES
         + /S1*/S2* S3*/S4                            ;BETWEEN HIGH AND LOW WITH
         + /S1* S2*/S3*/S4* X4                        ;HALF OR QUARTER THE
         +  S1*/S2* S3* S4* X4                        ;CLK FREQUENCY
         +  S1* S2* S3*/S4
         + /S1*/S2*/S3* S4
         + /S1* S2* S3* S4* X4
         +  S1*/S2*/S3*/S4* X4

/UD     := /S1* S2*/S3* S4                            ;THIS OUTPUT DETERMINES
         + /S1* S2* S3* S4                            ;IF SIGNAL PHI0 LEADS
         + /S1* S2* S3*/S4                            ;OR LAGS SIGNAL PHI90
         +  S1* S2* S3*/S4
         +  S1*/S2* S3*/S4
         +  S1*/S2*/S3*/S4
         +  S1*/S2*/S3* S4
         + /S1*/S2*/S3* S4
```

DESCRIPTION

THIS PAL16R4 IMPLEMENTS A TWO CHANNEL SHAFT ENCODER OF THE TYPE USED IN SPEED CONTROLLERS AND OPTICAL DEVICES.

BOTH THE "UP" AND "DOWN" OUTPUTS OF THE PAL ARE NORMALLY HIGH.

WHEN THE SIGNAL AT THE "PHI0" INPUT LEADS THE SIGNAL AT THE "PHI90" INPUT, THE "DOWN" OUTPUT ALTERNATES BETWEEN HIGH AND LOW LEVELS AT HALF THE "CLK" FREQUENCY RATE. ALSO, WHEN THE SIGNAL AT THE "PHI0" INPUT LAGS THE SIGNAL AT THE "PHI90" INPUT, THE "UP" OUTPUT ALTERNATES BETWEEN HIGH AND LOW LEVELS AT HALF THE "CLK" FREQUENCY RATE.

THE SHAFT ENCODER FEATURES THE CONFIGURATION AND OUTPUT POLARITY TO DRIVE AN 74S193 TYPE UP/DOWN COUNTER.

THIS DESIGN WITH GLITCHFREE OUTPUTS WILL BE EXTREMELY USEFUL IN ELECTRICALLY NOISY ENVIRONMENTS. THE PINNING IS GIVEN AS A FIRST PROPOSAL AND CAN BE CHANGED ACCORDING TO THE PC-BOARD LAYOUT.

```
FUNCTION TABLE

CLK /OC /SET PHI0 PHI90 X4 S1 S2 S3 S4 COUNT UD

;---CONTROLS---      --INPUTS--     X      SSSS     -OUTPUTS-
;CLK  /OC  /SET      PHI0 PHI90     4      1234     COUNT  UD       COMMENTS
-------------------------------------------------------------------------------
  C    L    L         X     X       X      LLLL       H     H       CLEAR REGISTERS
  C    L    H         L     L       L      LHLH       H     H       COUNT UP X4=L
  C    L    H         H     L       L      HHLH       H     L
  C    L    H         H     L       L      HLLH       L     H
  C    L    H         H     H       L      HLHH       H     L
  C    L    H         H     H       L      HLHL       H     H
  C    L    H         L     H       L      LLHL       H     L
  C    L    H         L     H       L      LHHL       L     H
  C    L    H         L     L       L      LHLL       H     L
  C    L    H         L     L       L      LHLH       H     H
  C    L    H         H     L       L      HHLH       H     L
  C    L    H         H     L       L      HLLH       L     H
  C    L    H         H     H       L      HLHH       H     L
  C    L    H         H     H       L      HLHL       H     H
  C    L    H         L     H       L      LLHL       H     L
  C    L    H         L     H       L      LHHL       L     H
  C    L    H         L     L       H      LHLL       H     L       COUNT UP X4=L
  C    L    H         L     L       H      LHLH       L     H
  C    L    H         H     L       H      HHLH       H     L
  C    L    H         H     L       H      HLLH       L     H
  C    L    H         H     H       H      HLHH       H     L
  C    L    H         H     H       H      HLHL       L     H
  C    L    H         L     H       H      LLHL       H     L
  C    L    H         L     H       H      LHHL       L     H
  C    L    H         L     L       H      LHLL       H     L
  C    L    H         L     L       H      LHLH       L     H
  C    L    H         H     L       H      HHLH       H     L
  C    L    H         H     L       H      HLLH       L     H
  C    L    H         H     H       H      HLHH       H     L
  C    L    H         H     H       H      HLHL       L     H
  C    L    L         X     X       X      LLLL       H     L       CLEAR REGISTERS
  C    L    H         L     L       L      LHLH       H     H       COUNT DOWN X4=L
  C    L    H         L     H       L      LHHH       H     L
  C    L    H         L     H       L      LHHL       H     L
  C    L    H         H     H       L      HHHL       H     L
  C    L    H         H     H       L      HLHL       L     L
  C    L    H         H     L       L      HLLL       H     L
  C    L    H         H     L       L      HLLH       H     L
  C    L    H         L     L       L      LLLH       H     L
  C    L    H         L     L       L      LHLH       L     L
  C    L    H         L     H       L      LHHH       H     L
  C    L    H         L     H       L      LHHL       H     L
  C    L    H         H     H       L      HHHL       H     L
  C    L    H         H     H       L      HLHL       L     L
  C    L    H         H     L       L      HLLL       H     L
  C    L    H         H     L       L      HLLH       H     L
  C    L    H         L     L       H      LLLH       H     L       COUNT DOWN X4=H
  C    L    H         L     L       H      LHLH       L     L
```

```
;---CONTROLS---      --INPUTS--     X      SSSS     -OUTPUTS-
;CLK  /OC  /SET      PHI0 PHI90     4      1234     COUNT  UD        COMMENTS
------------------------------------------------------------------------------
  C    L    H         L     H       H      LHHH       H    L
  C    L    H         L     H       H      LHHL       L    L
  C    L    H         H     H       H      HHHL       H    L
  C    L    H         H     H       H      HLHL       L    L
  C    L    H         H     L       H      HLLL       H    L
  C    L    H         H     L       H      HLLH       L    L
  C    L    H         L     L       H      LLLH       H    L
  C    L    H         L     L       H      LHLH       L    L
  C    L    H         L     H       H      LHHH       H    L
  C    L    H         L     H       H      LHHL       L    L
  C    L    H         H     H       H      HHHL       H    L
  C    L    H         H     H       H      HLHL       L    L
  X    H    X         X     X       X      ZZZZ       Z    Z         TEST HI-Z
------------------------------------------------------------------------------
```

DESCRIPTION

THIS PAL16R8 IMPLEMENTS A TWO CHANNEL SHAFT ENCODER OF THE TYPE USED IN SPEED CONTROLLERS AND OPTICAL DEVICES.

THE "COUNT" OUTPUT OF THE PAL IS NORMALLY HIGH. DURING SHAFT ENCODING THIS OUTPUT ALTERNATES BETWEEN HIGH AND LOW.

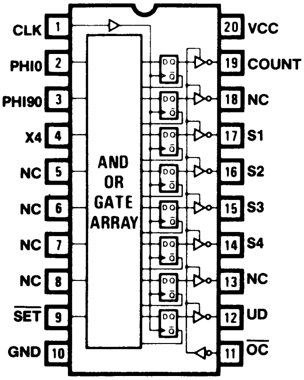
INPUT "X4" SELECTS BETWEEN HALF (X4=H) OR QUARTER (X4=L) CLK FREQUENCY OF THE "COUNTER" OUTPUT.

OUTPUT "UD" DETERMINES WHETHER SIGNAL PHI0 LEADS (UD=H) OR LAGS (UD=L) SIGNAL PHI90.

THE SHAFT ENCODER FEATURES THE CONFIGURATION AND OUTPUT POLARITY TO DRIVE AN 74S697 TYPE UP/DOWN COUNTER.

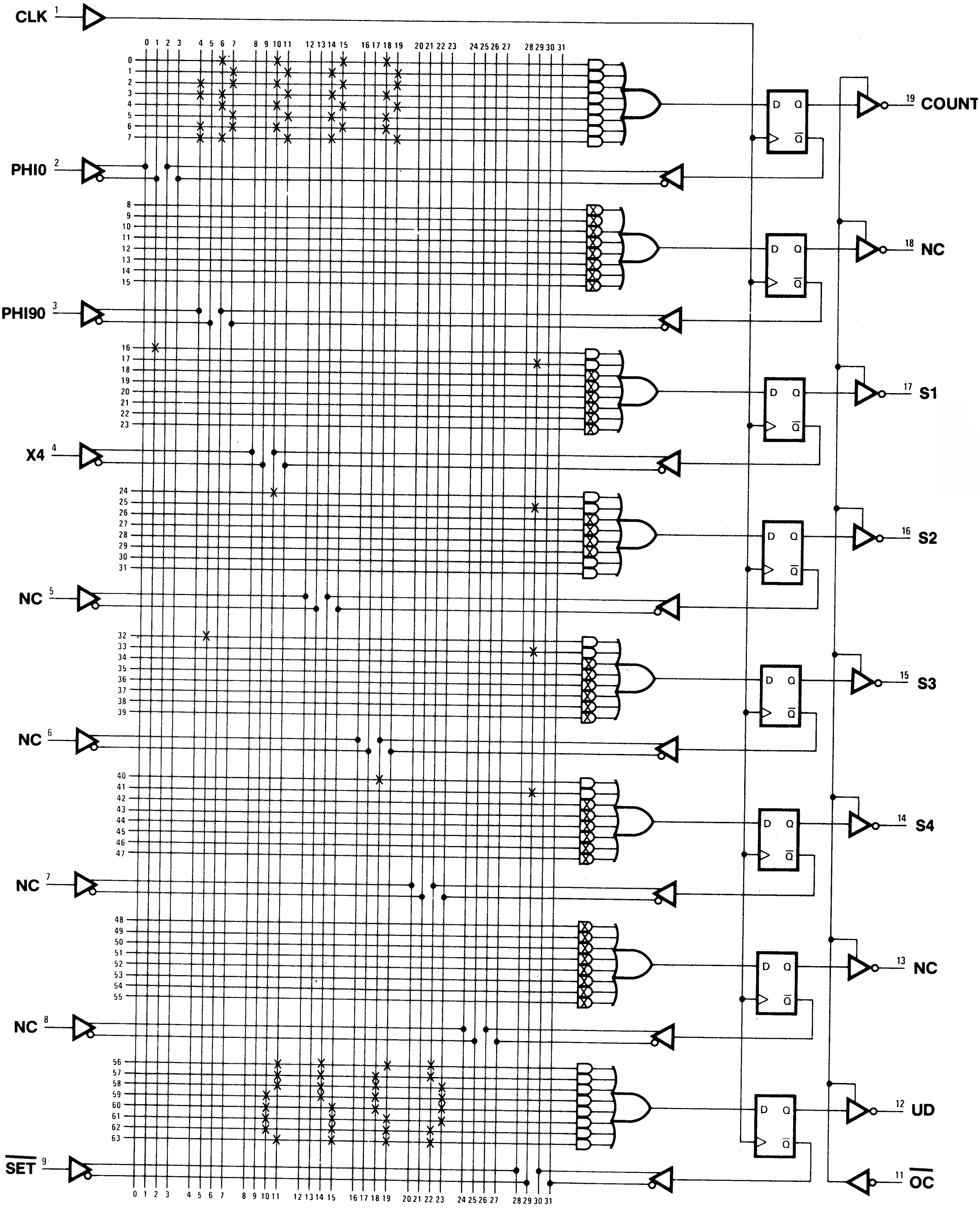
THIS DESIGN WITH GLITCHFREE OUTPUTS WILL BE EXTREMELY USEFUL IN ELECTRICALLY NOISY ENVIRONMENTS. THE PINNING IS GIVEN AS A FIRST PROPOSAL AND CAN BE CHANGED ACCORDING TO THE PC-BOARD LAYOUT.

PAL16R8



Shaft Encoder No. 2

Logic Diagram PAL16R8



```
PAL20X10                                                 PAL DESIGN SPECIFICATION
P7096                                                      WILLY VOLDAN 09/09/82
SHAFT ENCODER No. 3 (WITH INTERNAL 4-BIT UP/DOWN COUNTER)
MMI GMBH MUNICH
CLK PHI0 PHI90 X4 /LD NC D3 D2 D1 D0 /SET GND
/OC DOWN S4 S3 S2 S1 Q3 Q2 Q1 Q0 UP VCC

  /S1 :=  /PHI0                              ;CHECK FOR PHI0
      +    SET                               ;INITIALIZE S1=L

  /S2 :=  /S1                                ;CHECK FOR S1
      +    SET                               ;INITIALIZE S2=L

  /S3 :=  /PHI90                             ;CHECK FOR PHI90
      +    SET                               ;INITIALIZE S3=L

  /S4 :=  /S3                                ;CHECK FOR S3
      +    SET                               ;INITIALIZE S4=L

/DOWN :=    S1* S2* S3*/S4* PHI0* PHI90* X4  ;PHI0 LEADS PHI90 - COUNT=FREQ/2
      +   /S1*/S2*/S3* S4*/PHI0*/PHI90* X4   ;PHI0 LEADS PHI90 - COUNT=FREQ/2
     :+:    S1*/S2*/S3*/S4* PHI0*/PHI90      ;PHI0 LEADS PHI90 - COUNT=FREQ/4
      +   /S1* S2* S3* S4*/PHI0* PHI90       ;PHI0 LEADS PHI90 - COUNT=FREQ/4

/UP   :=  /S1*/S2* S3*/S4*/PHI0* PHI90       ;PHI90 LEADS PHI0 - COUNT=FREQ/4
      +    S1* S2*/S3* S4* PHI0*/PHI90       ;PHI90 LEADS PHI0 - COUNT=FREQ/4
     :+:   S1*/S2* S3* S4* PHI0* PHI90* X4   ;PHI90 LEADS PHI0 - COUNT=FREQ/2
      +   /S1* S2*/S3*/S4*/PHI0*/PHI90* X4   ;PHI90 LEADS PHI0 - COUNT=FREQ/2

/Q0   :=  /SET* LD*/D0                       ;LOAD D0 (LSB)
      +   /SET*/LD*/Q0                       ;HOLD Q0
     :+:  /SET*/LD* UP*/DOWN                 ;DECREMENT
      +   /SET*/LD*/UP* DOWN                 ;INCREMENT

/Q1   :=  /SET* LD*/D1                       ;LOAD D1
      +   /SET*/LD*/Q1                       ;HOLD Q1
     :+:  /SET*/LD* UP*/DOWN*/Q0             ;DECREMENT
      +   /SET*/LD*/UP* DOWN* Q0             ;INCREMENT

/Q2   :=  /SET* LD*/D2                       ;LOAD D2
      +   /SET*/LD*/Q2                       ;HOLD Q2
     :+:  /SET*/LD* UP*/DOWN*/Q0*/Q1         ;DECREMENT
      +   /SET*/LD*/UP* DOWN* Q0* Q1         ;INCREMENT

/Q3   :=  /SET* LD*/D3                       ;LOAD D3 (MSB)
      +   /SET*/LD*/Q3                       ;HOLD Q3
     :+:  /SET*/LD* UP*/DOWN*/Q0*/Q1*/Q2     ;DECREMENT
      +   /SET*/LD*/UP* DOWN* Q0* Q1* Q2     ;INCREMENT
```

```
FUNCTION TABLE

CLK /OC /SET /LD X4 PHI0 PHI90 S1 S2 S3 S4 UP DOWN D3 D2 D1 D0 Q3 Q2 Q1 Q0

;----CONTROLS----  INPUT
;    /  /   /        PHI    SSSS             DDDD  QQQQ     COMMENTS
;CLK OC SET LD X4   0  90   1234  UP DOWN   3210  3210   (Q HEX VALUE)
-------------------------------------------------------------------------------
  C   L  L   X  X    X X    LLLL  H    H    XXXX  HHHH   INITIALIZE REGISTERS (F)
  C   L  H   L  X    X X    XXXX  X    X    HLHL  HLHL   LOAD (A)
  C   L  H   H  L    L L    LLLL  H    H    XXXX  HLHL   HOLD (A)
  C   L  H   H  L    H L    HLLL  H    H    XXXX  HLHL   HOLD (A) PHI0 LEADS PHI90
  C   L  H   H  L    H L    HHLL  H    L    XXXX  HLHL   HOLD (A) X4=H - FREQ/4
  C   L  H   H  L    H H    HHHL  H    H    XXXX  HLLH   DECREMENT (9)
  C   L  H   H  L    H H    HHHH  H    H    XXXX  HLLH   HOLD (9)
  C   L  H   H  L    L H    LHHH  H    H    XXXX  HLLH   HOLD (9)
  C   L  H   H  L    L H    LLHH  H    L    XXXX  HLLH   HOLD (9)
  C   L  H   H  L    L L    LLLH  H    H    XXXX  HLLL   DECREMENT (8)
  C   L  H   H  L    L L    LLLL  H    H    XXXX  HLLL   HOLD (8)
  C   L  H   H  L    H L    HLLL  H    H    XXXX  HLLL   HOLD (8)
  C   L  H   H  L    H L    HHLL  H    L    XXXX  HLLL   HOLD (8)
  C   L  H   H  L    H H    HHHL  H    H    XXXX  LHHH   DECREMENT (7)
  C   L  H   H  L    H H    HHHH  H    H    XXXX  LHHH   HOLD (7)
  C   L  H   H  L    L H    LHHH  H    H    XXXX  LHHH   HOLD (7)
  C   L  H   H  L    L H    LLHH  H    L    XXXX  LHHH   HOLD (7)
  C   L  H   H  H    L L    LLLH  H    H    XXXX  LHHL   DECREMENT (6)
  C   L  H   H  H    L L    LLLL  H    L    XXXX  LHHL   HOLD (6) X4=H - FREQ/2
  C   L  H   H  H    H L    HLLL  H    H    XXXX  LHLH   DECREMENT (5)
  C   L  H   H  H    H L    HHLL  H    L    XXXX  LHLH   HOLD (5)
  C   L  H   H  H    H H    HHHL  H    H    XXXX  LHLL   DECREMENT (4)
  C   L  H   H  H    H H    HHHH  H    L    XXXX  LHLL   HOLD (4)
  C   L  H   H  H    L H    LHHH  H    H    XXXX  LLHH   DECREMENT (3)
  C   L  H   H  H    L H    LLHH  H    L    XXXX  LLHH   HOLD (3)
  C   L  H   H  H    L L    LLLH  H    H    XXXX  LLHL   DECREMENT (2)
  C   L  H   H  H    L L    LLLL  H    L    XXXX  LLHL   HOLD (2)
  C   L  H   H  H    H L    HLLL  H    H    XXXX  LLLH   DECREMENT (1)
  C   L  H   H  H    H L    HHLL  H    L    XXXX  LLLH   HOLD (1)
  C   L  H   H  H    H H    HHHL  H    H    XXXX  LLLL   DECREMENT (0)
  C   L  H   H  H    H H    HHHH  H    L    XXXX  LLLL   HOLD (0)
  C   L  H   H  H    L H    LHHH  H    H    XXXX  HHHH   DECREMENT (F)(ROLL UNDER)
  C   L  H   H  H    L H    LLHH  H    L    XXXX  HHHH   HOLD (F)
  C   L  H   H  H    L L    LLLH  H    H    XXXX  HHHL   DECREMENT (E)
  C   L  H   L  X    X X    XXXX  X    X    LHLH  LHLH   LOAD (5)
  C   L  H   H  L    L L    LLLL  H    H    XXXX  LHLH   HOLD (5)
  C   L  H   H  L    L H    LLHL  H    H    XXXX  LHLH   HOLD (5) PHI90 LEADS PHI0
  C   L  H   H  L    L H    LLHH  L    H    XXXX  LHLH   HOLD (5) X4=L - FREQ/4
  C   L  H   H  L    H H    HLHH  H    H    XXXX  LHHL   INCREMENT (6)
  C   L  H   H  L    H H    HHHH  H    H    XXXX  LHHL   HOLD (6)
  C   L  H   H  L    H L    HHLH  H    H    XXXX  LHHL   HOLD (6)
  C   L  H   H  L    H L    HHLL  L    H    XXXX  LHHL   HOLD (6)
  C   L  H   H  L    L L    LHLL  H    H    XXXX  LHHH   INCREMENT (7)
  C   L  H   H  L    L L    LLLL  H    H    XXXX  LHHH   HOLD (7)
  C   L  H   H  L    L H    LLHL  H    H    XXXX  LHHH   HOLD (7)
  C   L  H   H  L    L H    LLHH  L    H    XXXX  LHHH   HOLD (7)
  C   L  H   H  L    H H    HLHH  H    H    XXXX  HLLL   INCREMENT (8)
  C   L  H   H  L    H H    HHHH  H    H    XXXX  HLLL   HOLD (8)
  C   L  H   H  L    H L    HHLH  H    H    XXXX  HLLL   HOLD (8)
```

```
;----CONTROLS----  INPUT
;    /  /   /       PHI  SSSS          DDDD  QQQQ     COMMENTS
;CLK OC SET LD X4  0  90 1234 UP DOWN  3210  3210   (Q HEX VALUE)
-------------------------------------------------------------------------------
  C   L  H   H  L   H L  HHLL  L   H   XXXX  HLLL   HOLD (8)
  C   L  H   H  H   L L  LHLL  H   H   XXXX  HLLH   INCREMENT (9)
  C   L  H   H  H   L L  LLLL  L   H   XXXX  HLLH   HOLD (9)  X4=H - FREQ/2
  C   L  H   H  H   L H  LLHL  H   H   XXXX  HLHL   INCREMENT (A)
  C   L  H   H  H   L H  LLHH  L   H   XXXX  HLHL   HOLD (A)
  C   L  H   H  H   H H  HLHH  H   H   XXXX  HLHH   INCREMENT (B)
  C   L  H   H  H   H H  HHHH  L   H   XXXX  HLHH   HOLD (B)
  C   L  H   H  H   H L  HHLH  H   H   XXXX  HHLL   INCREMENT (C)
  C   L  H   H  H   H L  HHLL  L   H   XXXX  HHLL   HOLD (C)
  C   L  H   H  H   L L  LHLL  H   H   XXXX  HHLH   INCREMENT (D)
  C   L  H   H  H   L L  LLLL  L   H   XXXX  HHLH   HOLD (D)
  C   L  H   H  H   L H  LLHL  H   H   XXXX  HHHL   INCREMENT (E)
  C   L  H   H  H   L H  LLHH  L   H   XXXX  HHHL   HOLD (E)
  C   L  H   H  H   H H  HLHH  H   H   XXXX  HHHH   INCREMENT (F)
  C   L  H   H  H   H H  HHHH  L   H   XXXX  HHHH   HOLD (F)
  C   L  H   H  H   H L  HHLH  H   H   XXXX  LLLL   INCREMENT (0) (ROLL OVER)
  C   L  H   H  H   H L  HHLL  L   H   XXXX  LLLL   HOLD (0)
  C   L  H   H  H   L L  LHLL  H   H   XXXX  LLLH   INCREMENT (1)
  C   L  H   L  X   X X  XXXX  X   X   LLLH  LLLH   LOAD (1)
  C   L  H   L  X   X X  XXXX  X   X   LLHH  LLHH   LOAD (3)
  C   L  H   L  X   X X  XXXX  X   X   LHLH  LHLH   LOAD (5)
  C   L  H   L  X   X X  XXXX  X   X   LHHH  LHHH   LOAD (7)
  C   L  H   L  X   X X  XXXX  X   X   HLLH  HLLH   LOAD (9)
  C   L  H   L  X   X X  XXXX  X   X   HLHH  HLHH   LOAD (B)
  C   L  H   L  X   X X  XXXX  X   X   HHLH  HHLH   LOAD (D)
  C   L  H   L  X   X X  XXXX  X   X   HHHH  HHHH   LOAD (F)
  X   H  X   X  X   X X  ZZZZ  Z   Z   XXXX  ZZZZ   TEST HI-Z
-------------------------------------------------------------------------------
```

DESCRIPTION

THIS PAL20X10 IMPLEMENTS A TWO CHANNEL SHAFT ENCODER WITH AN INTERNAL 4-BIT UP/DOWN COUNTER.

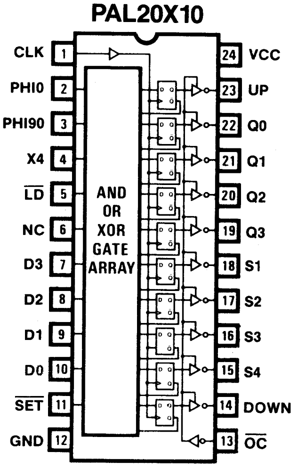
BOTH THE "UP" AND "DOWN" OUTPUTS OF THE PAL ARE NORMALLY AT HIGH.

WHEN THE SIGNAL AT THE "PHI0" INPUT LEADS THE SIGNAL AT THE "PHI90" INPUT, THE "DOWN" OUTPUT ALTERNATES BETWEEN HIGH AND LOW LEVELS AND THE COUNTER WILL COUNT DOWN. WHEN THE SIGNAL AT THE "PHI0" INPUT LEADS THE SIGNAL AT THE "PHI90" INPUT, THE "UP" OUTPUT ALTERNATES BETWEEN HIGH AND LOW LEVELS AND THE COUNTER WILL COUNT UP.

INPUT "X4" SELECTS BETWEEN HALF (X4=H) OR QUARTER (X4=L) CLK FREQUENCY OF THE COUNTER OUTPUTS.

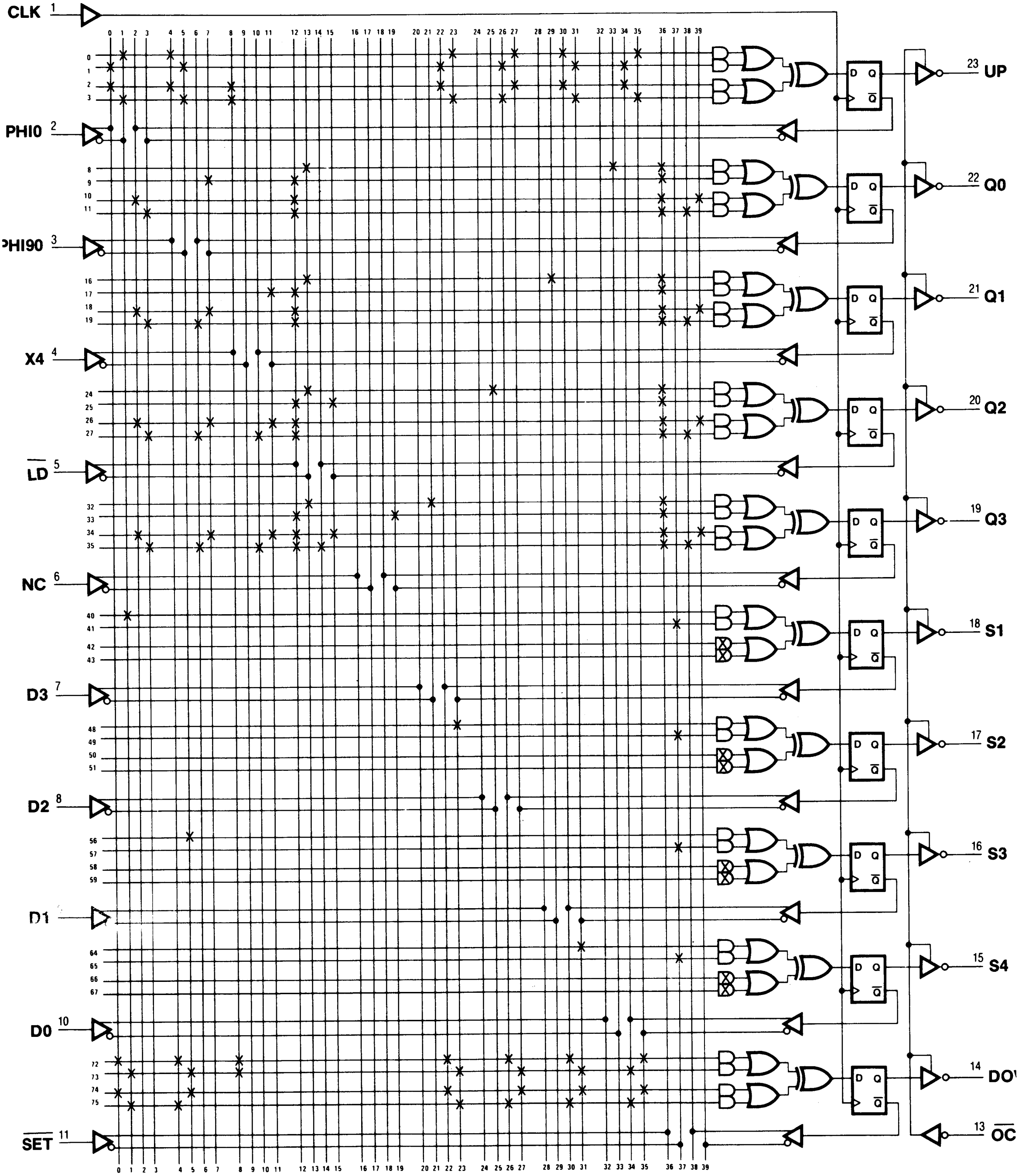
THE INTERNAL 4-BIT SYNCHRONOUS COUNTER HAS COUNT-UP, COUNT DOWN CAPABILITIES. ALSO, THE COUNTER CAN PARALLEL LOAD AND HOLD DATA INDEPENDENTLY OF THE SHAFT ENCODER SECTION. THE REGISTERS ARE SYNCHRONOUSLY INITIALIZED WHEN /SET IS HELD LOW.

THE CONTROL INPUTS PROVIDE THESE OPERATIONS WHICH OCCUR SYNCHRONOUSLY AT THE RISING EDGE OF THE CLOCK.



Shaft Encoder No. 3
(With Internal 4-Bit Up/Down Counter)

Logic Diagram PAL20X10



Stepper Motor Controller

Dave Sackett

Stepper motors and linear actuators are used in a variety of applications requiring precise rotational and/or linear movement. Examples are printers, floppy disk drives, mechanical valves, etc. Stepper motors are two-phase permanent magnet motors which provide discrete angular movement every time the polarity of a winding is changed.

The control and drive circuitry for such a motor can be implemented digitally by using a PAL with high current drive buffering on the output. The details of such a design are detailed here.

The explosion of activity in fault-tolerant systems has provoked renewed interest in Error Detection and Correction (EDC) techniques. Of the several approaches available, one of the simplest is the Hamming Code. Hamming codes work by introducing redundancy bits in a parallel data word. This application discusses the implementation of such a system using PAL devices.

2175 Mission College Boulevard, Santa Clara, CA 95050 Tel: (408) 970-9700 TWX: 910-338-2376
TWX: 910-338-2374

Monolithic Memories MMI

Functional Description

Stepper motors and linear actuators are used in a variety of applications requiring precise rotational and/or linear movement. Examples are printers, floppy disc drives, mechanical valves, etc. Stepper motors are two-phase permanent magnet motors which provide discrete angular movement every time the polarity of a winding is changed. In the case of linear actuators, the angular movement is converted to a linear movement via a load screw. In essence, they are dc motors without brushes, where the user provides commutation with external logic.

Circuit Operation

One type of drive circuit, unipolar drive, is shown in Figure 1 below. Two drive sequences are given in Tables 1A and 1B. Angular rotation is achieved by saturating the transistor drivers in the sequence shown in the appropriate table (full or half step). Now, assume the circuit of Figure 1 is connected to a stepper motor designed for 7.5° steps. By following the step sequence of Table 1A (full step), the shaft will rotate 7.5° each time the state is changed. If the sequence of Table 1B is followed, a 3.75° (half step) rotation will result for each change of state. For both step sequences, the direction can be reversed by stepping backwards through the table (step 4-3-2-1-4-etc.).

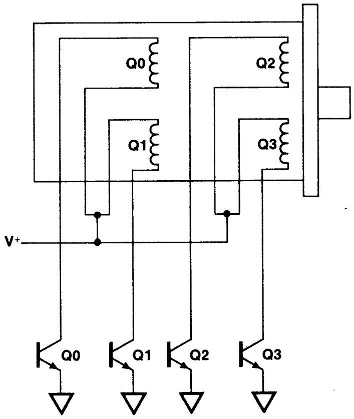


Figure 1

Table 1A
FULL STEP SEQUENCE

CLOCKWISE ROTATION ↓ / ↑ COUNTER-CLOCKWISE ROTATION

STEP	Q0	Q1	Q2	Q3
1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	1	0
1	1	0	1	0

Table 1B
HALF STEP SEQUENCE

CLOCKWISE ROTATION ↓ / ↑ COUNTER-CLOCKWISE ROTATION

STEP	Q0	Q1	Q2	Q3
1	1	0	1	0
2	1	0	0	0
3	1	0	0	1
4	0	0	0	1
5	0	1	0	1
6	0	.1	0	0
7	0	1	1	0
8	0	0	1	0
1	1	0	1	0

PAL Implementation

In this application, one PAL16R4 can be used to provide the logic levels required to drive two stepper motors in the full step mode. Due to the high current drive required (100-400 mA/phase), external inverting high current buffers would be used (ULN 2001 or equivalent). In the design, the following features are provided within the PAL:

- Enable/Disable inputs to enable stepping of either section. (/E inputs).
- Select clockwise or counter-clockwise rotation.
- Set the motor to logic state step 1.

A block diagram/pinout is shown in Figure 2.

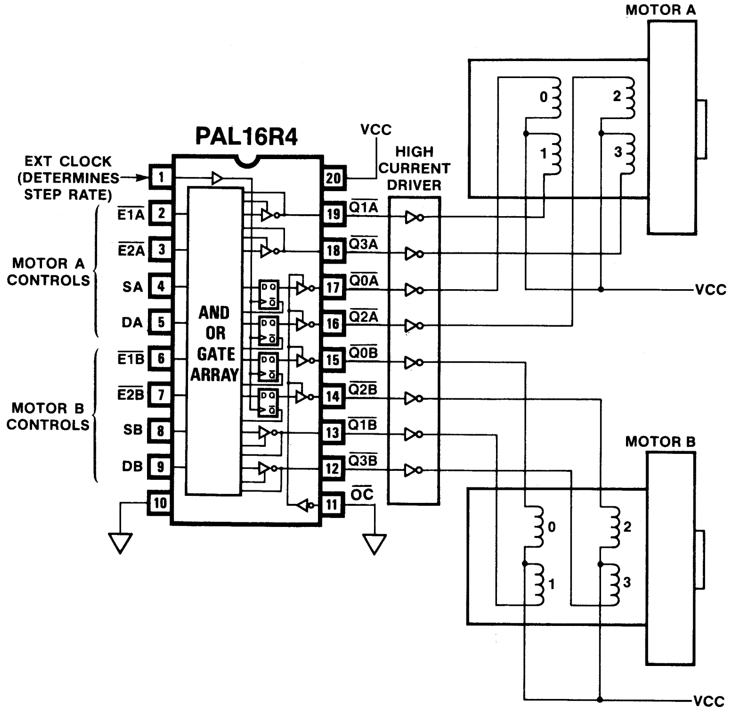


Figure 2

A function table for each motor control section is given below.

CLOCK	$\overline{E1}$	$\overline{E2}$	S	D	FUNCTION
X	1	X	X	X	Hold motor in current position
X	X	1	X	X	Hold motor in current position
↑	0	0	1	X	Set outputs to step 1 levels
↑	0	0	0	0	Step motor clockwise
↑	0	0	0	1	Step motor counter-clockwise

The full step sequence (Table 1A) can be simplified from 4 outputs to 2 outputs since Q1 = Q0 and Q3 = Q2. The sequences can then be expressed as follows:

D = 0			D = 1	
STEP	Q0	Q2	Q0	Q2
1	1	1	1	1
2	1	0	0	1
3	0	0	0	0
4	0	1	1	0
1	1	1	1	1

Q1 = $\overline{Q0}$ Q3 = $\overline{Q2}$

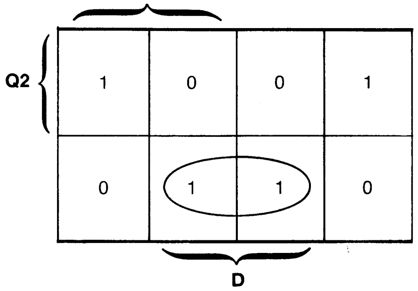
when S = 1:

Q0 = 1 Q1 = 0 Q2 = 1 Q3 = 0

when E1 = 1 or E2 = 1

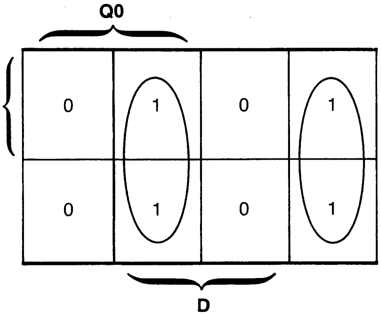
Q0 = q0 Q1 = q1 Q2 = q2 Q3 = q3

The step sequences can be converted to equations by use of a Karnaugh map.



1	0	0	1
0	1	1	0

$Q0 = Q2 \cdot \overline{D} + \overline{Q2} \cdot D$



0	1	0	1
0	1	0	1

$Q2 = Q0 \cdot D + \overline{Q0} \cdot \overline{D}$

Factor in E1 and E2:

$Q0 = \overline{E1} \cdot \overline{E2} \cdot Q2 \cdot \overline{D} + \overline{E1} \cdot \overline{E2} \cdot \overline{Q2} \cdot \overline{D}$

$Q2 = \overline{E1} \cdot \overline{E2} \cdot Q0 \cdot D + \overline{E1} \cdot \overline{E2} \cdot \overline{Q0} \cdot \overline{D}$

Express the set function as an equation:

$Q0 = \overline{E1} \cdot E2 \cdot S \qquad Q2 = \overline{E1} \cdot \overline{E2} \cdot S$

Express the hold function (when E1 or E2 = 1)

$Q0 = q0 \cdot E1 + q0 \cdot E2 \quad Q2 = q2 \cdot E1 + q2 \cdot E2$

Combining all the above:

$Q0 := \overline{E1} \cdot \overline{E2} \cdot S + Q0 \cdot E1 + Q0 \cdot E2 + \overline{E1} \cdot \overline{E2} \cdot Q2 \cdot \overline{D} + \overline{E1} \cdot \overline{E2} \cdot \overline{Q2} \cdot D$

$Q1 := Q0$

$Q2 := \overline{E1} \cdot \overline{E2} \cdot S + Q2 \cdot E1 + Q2 \cdot E2 + \overline{E1} \cdot \overline{E2} \cdot Q0 \cdot D + \overline{E1} \cdot \overline{E2} \cdot \overline{Q0} \cdot \overline{D}$

$Q3 := Q2$

Conclusion

Although this example could be used "as is" in a stepper motor application, the programmability of PAL device could allow for any desired modifications. Changes to the circuit might include:

1. Drive only one stepper motor, using a PAL16R6. The other flip-flops could be used as a programmable counter, allowing for different speed settings.
2. Drive only one stepper motor, using the extra inputs and outputs to handle other circuit functions.
3. Drive only one stepper motor, using a PAL16R6. The other flip-flops could be used as a 4-bit position counter.
4. The substitution of a PAL16R8, and another *inverting* buffer would allow the driving and control of four stepper motors.
5. Re-program for half-step operation.

```
PAL16R4                                               PAL DESIGN SPECIFICATION
SMC                                                     DAVE SACKETT 02/23/81
STEPPER MOTOR CONTROLLER
DEVOE COMPANY, INDIANAPOLIS, INDIANA
CLK /E1A /E2A SA DA /E1B /E2B SB DB GND
/OC /Q3B /Q1B /Q2B /Q0B /Q2A /Q0A /Q3A /Q1A VCC


Q0A :=  Q0A*/E1A                    ;HOLD IF NOT E1
     +  Q0A*/E2A                    ;HOLD IF NOT E2
     +  SA * E1A* E2A               ;STEP 1 IF SET
     + /Q2A* E1A* E2A* DA           ;LOAD /Q2A IF COUNTER-CLOCKWISE
     +  Q2A* E1A* E2A*/DA           ;LOAD  Q2A IF CLOCKWISE

IF (VCC) Q1A = /Q0A

Q2A :=  Q2A*/E1A                    ;HOLD IF NOT E1
     +  Q2A*/E2A                    ;HOLD IF NOT E2
     +  SA * E1A* E2A               ;STEP 1 IF SET
     +  Q0A* E1A* E2A* DA           ;LOAD  Q0A IF COUNTER-CLOCKWISE
     + /Q0A* E1A* E2A*/DA           ;LOAD /Q0A IF CLOCKWISE

IF (VCC) Q3A = /Q2A

Q0B :=  Q0B*/E1B                    ;HOLD IF NOT E1
     +  Q0B*/E2B                    ;HOLD IF NOT E2
     +  SB * E1B* E2B               ;STEP 1 IF SET
     + /Q2B* E1B* E2B* DB           ;LOAD /Q2B IF COUNTER-CLOCKWISE
     +  Q2B* E1B* E2B*/DB           ;LOAD  Q2B IF CLOCKWISE

IF (VCC) Q1B = /Q0B

Q2B :=  Q2B*/E1B                    ;HOLD IF NOT E1
     +  Q2B*/E2B                    ;HOLD IF NOT E2
     +  SB * E1B* E2B               ;STEP 1 IF SET
     +  Q0B* E1B* E2B* DB           ;LOAD  Q0B IF COUNTER-CLOCKWISE
     + /Q0B* E1B* E2B*/DB           ;LOAD /Q0B IF CLOCKWISE

IF (VCC) Q3B = /Q2B
```

```
FUNCTION TABLE

CLK /OC /E1A /E2A SA DA Q0A Q1A Q2A Q3A /E1B /E2B SB DB Q0B Q1B Q2B Q3B

;CHIP      STEPPER MOTOR A    STEPPER MOTOR B
;C  /       CONTROL  STEP      CONTROL  STEP
;L  O       E E S D  QQQQ      E E S D  QQQQ
;K  C       1 2 A A  0123      1 2 A A  0123      COMMENTS
------------------------------------------------------------------
 C  L       L L H X  HLHL      L L H X  HLHL      SET TO STEP 1
 C  L       H H X X  HLHL      H H X X  HLHL      HOLD
 C  L       L L L L  HLLH      L L L H  LHHL      STEP A CW, B CCW
 C  L       L L L L  LHLH      L L L H  LHLH      STEP A CW, B CCW
 C  L       L L L L  LHHL      L L L H  HLLH      STEP A CW, B CCW
 C  L       L L L L  HLHL      L L L H  HLHL      STEP A CW, B CCW
 C  L       L L L L  HLLH      L L L H  LHHL      STEP A CW, B CCW
 C  L       L L L L  LHLH      L L L H  LHLH      STEP A CW, B CCW
 C  L       L L L H  HLLH      L L L L  LHHL      STEP A CCW, B CW
 C  L       H L L H  HLLH      H L L L  LHHL      HOLD
 C  L       L H L H  HLLH      L H L L  LHHL      HOLD
 C  L       L H H H  HLLH      L H H L  LHHL      HOLD
------------------------------------------------------------------
```

DESCRIPTION

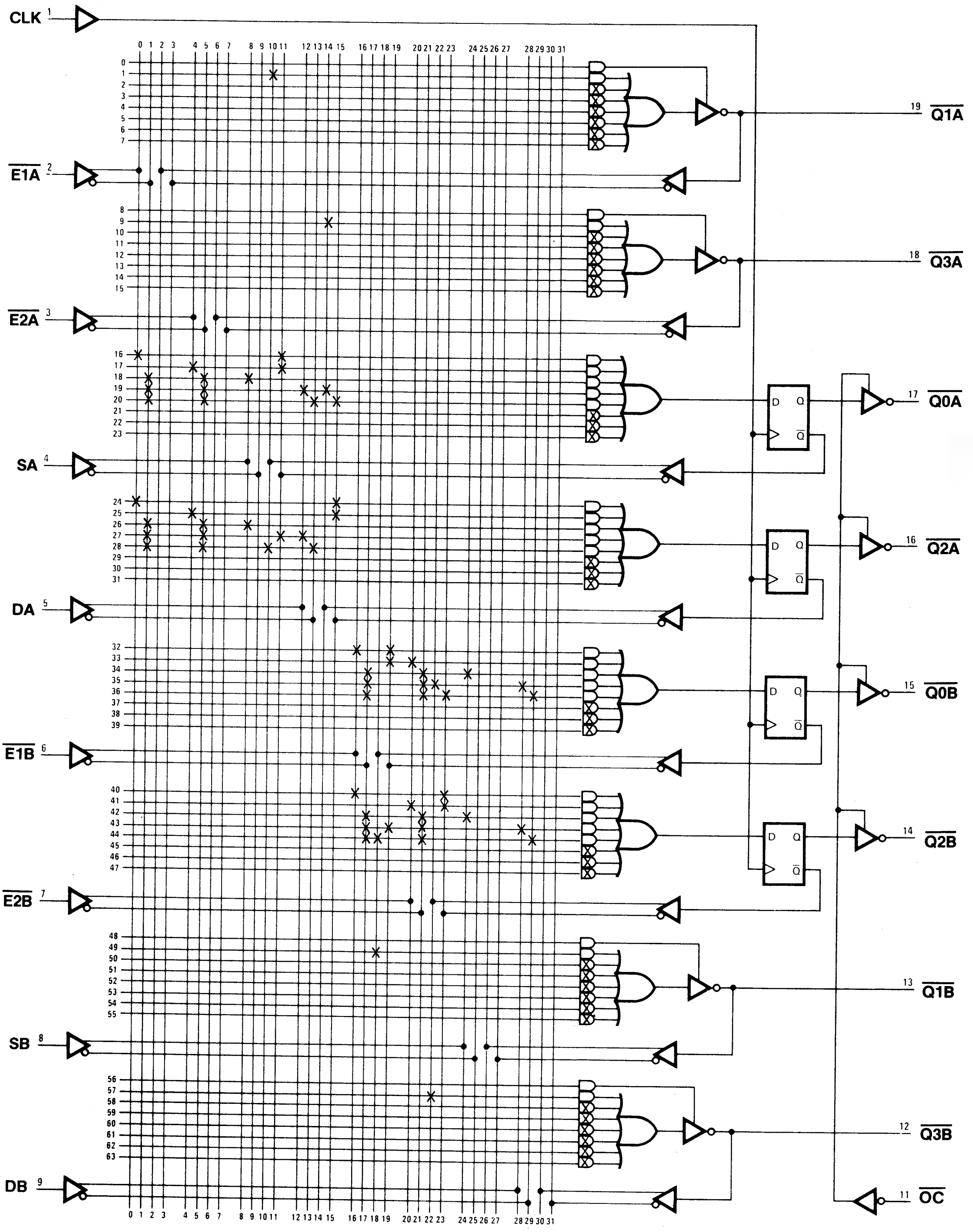
THIS PAL16R4 PROVIDES THE LOGIC LEVELS REQUIRED TO DRIVE TWO STEPPER MOTORS IN THE FULL STEP MODE.

THE FOLLOWING OPERATIONS MAY BE PERFORMED FOR EACH STEPPER MOTOR CONTROLLER INDIVIDUALLY:

CLK	/E1	/E2	S	D	OPERATION
X	H	X	X	X	HOLD MOTOR IN CURRENT POSITION
X	X	H	X	X	HOLD MOTOR IN CURRENT POSITION
C	L	L	H	X	SET OUTPUTS TO STEP 1 LEVELS
C	L	L	L	L	STEP MOTOR CLOCKWISE
C	L	L	L	H	STEP MOTOR COUNTER-CLOCKWISE

Stepper Motor Controller

Logic Diagram PAL16R4



Notes

Improving Your Memory With 'S700-Family MOS Drivers*

Chuck Hastings and Suneel Rajpal

Abstract

Dynamic-MOS random-access-memory integrated circuits (DRAMs) are the basic components used today as building blocks for larger computer-memory systems. Even though using DRAMs may seem very straightforward, there are some major pitfalls which designers must avoid.

This application note discusses the circumstances which arise when designing DRAM-array memory boards, such as wiring-trace capacitance and inductance, signal reflections, voltage undershoot, and asymmetric driver-circuit output impedances. Great improvements over a naive design approach are possible by practicing more sophisticated printed-wiring layout techniques, and by using second generation dynamic-MOS drivers rather than first-generation high-current drivers.

The 'S700/730/731/734 8-bit buffers are second-generation parts having electrical and switching characteristics which are especially tailored to driving the distributed-capacitance loads presented by large DRAM arrays. The rationale behind these new parts is presented here, and specific applications are discussed: avoiding information loss from a-c power failure, and fast multiplexing of row addresses with column addresses using the complementary-enable 'S700/731 drivers.

* This paper is a slightly modified version of the paper by the same name which appeared in the **Conference Proceedings of the 8th West Coast Computer Faire,** 18-20 March 1983.

Improving Your Memory With 'S700-Family MOS Drivers

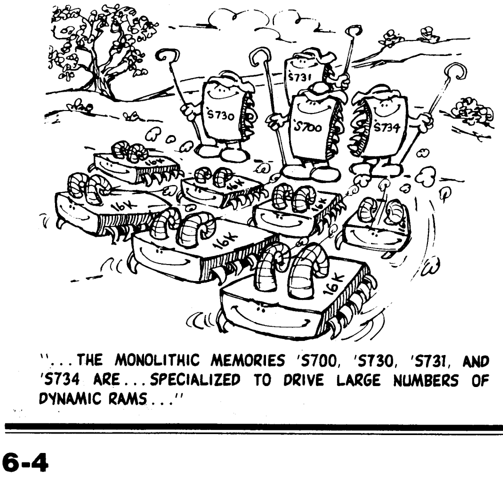
Chuck Hastings and Suneel Rajpal

Introduction

Today, fast-access-time high-density dynamic random-access-memory integrated circuits (DRAMs) are where it's at in the design of commercial computer memories of any size, from tabletop personal-computer memories to giant mainframe memories; magnetic cores are, now, "but a distant memory." As a computer-scene corollary to Parkinson's First Law[r1], "Work expands to fill the time available," it is observably **always** true that "**Computer software expands to fill the memory available.**" Thus, the rapid advancements which have been made in the cost, density, availability, second-source standardization, and reliability of DRAMs have generally come just in the nick of time to keep up with the computer industry's insatiable demand for ever-larger main memories. To pick but one example, the Hewlett-Packard 3000-series minicomputer family was originally introduced with a maximum main-memory configuration of 131,072 bytes; today, the maximum configuration is 8,388,608 bytes, and plans for even larger configurations are already taking shape.

Unfortunately, the technological advancements in the **peripheral** integrated circuits needed to drive all of these DRAMs have, to say the least, been noticeably less rapid. The usual design practice has been to drive large DRAM arrays with high-current buffers such as 'S240s, coupled with external series resistors in the driven signal lines. Now, with the introduction of the Monolithic Memories 'S700/730/731/734 MOS drivers, the memory designer's task is greatly simplified.

The 'S700, 'S730, 'S731, and 'S734 are fast and powerful Schottky-technology TTL 8-bit buffers, specialized to drive large numbers of dynamic RAMs. Their internal design is particularly well adapted to driving signal lines with lots and lots of **distributed capacitance.** They are **drop-in, pin-compatible replacements** for the respective first-generation 'S240-family high-current drivers — 'S210, 'S240, 'S241, and 'S244, which excel for their intended high-current applications or even for lumped-capacitance applications but can be awkward to use in typical DRAM memory-board designs.



"...THE MONOLITHIC MEMORIES 'S700, 'S730, 'S731, AND 'S734 ARE...SPECIALIZED TO DRIVE LARGE NUMBERS OF DYNAMIC RAMS..."

So that you understand the essentials of what you need to know to design memory boards which work, we'll first take a quick glance at the electrical situation, complete with equations. Don't worry — we won't actually **derive** these equations here; derivations are readily available in the literature[r2, r3], and our purpose is simply to motivate some otherwise arbitrary-sounding statements as to what constitutes good layout practice. Following that, we'll present the rationale behind the various members of the family and their differing functional behavior or "architecture." Finally, we'll discuss some pragmatic design issues; how to avoid information loss due to glitches in battery-backup-protected memory systems during power failure, and when and where to use the 'S700 and 'S731 complementary-enable parts.

The Memory-Board Design Problem

The central problem facing the designer of a memory board is to drive a large number of highly-capacitative DRAM address, data, and control inputs just as fast as they can safely be driven, since memory speed (like memory size) is something which computer-system designers can never get quite enough of. Typically, a designer places **from 70 to 300 DRAMs on a single board.** Now, the address and data inputs of a DRAM have very non-negligible input capacitances — 3.5 picofarads (pf) typical, and 5 or even 7 pf worst-case; the control inputs may have as much as 10 pf worst-case. Assuming 5 pf, the total capacitance per address or data line per board must by simple multiplication fall between 350 pf and 1500 pf — even more when the capacitance of the printed-circuit-board (PCB) wiring traces is reckoned with. These numbers are not at all the sort of numbers you normally see on the data sheets for most of the industry-standard 8-bit buffers — those have for many years conventionally been specified by all vendors at 15 pf, 50 pf, etc. apparently according to the proposition that "small is beautiful," i.e., the logic delays and waveforms come out more agreeably at those numbers.

In keeping with motherhood and apple pie, the memory-board design obviously must be optimized for speed, reliability, physical area, and dollar cost; the **topology** (the physical organization and length of the wiring traces) and the number of drivers are chosen accordingly. Since contemporary DRAMs receive their complete addresses in **two** pieces, a "row address" and a "column address" (corresponding to the cell layout within the DRAM chip), the speed of the address-driving circuits is **particularly** critical since the bit pattern transmitted on the address lines must be changed **twice** during each complete memory read or write cycle. In DRAM "architecture," the row and column addresses are of equal length, say n bits, and the width of the data word within the DRAM is one bit in most contemporary parts. The first DRAMs with this architecture, in the mid-1970s, had n = 6, and thus were 2^{12}x1 = 4096x1 or "4K" DRAMs. By now, of course, such tiny DRAM sizes are obsolete, and even 16K (16384x1) DRAMs are a super-low-cost commodity. Much commercial design today is being done with 64K (65536x1) DRAMs, and even larger DRAMs are coming soon; 256K (262144x1) DRAMs pin-compatible with the usual 64K types have been announced.

When all of these factors are taken into account, the practical upper limit to how many DRAM inputs can be hung on one trace is usually thought to be in the range of 80 to 100. This limit has some implications with respect to word length and word organization. The combined effect of the system word length as seen by the computer programmer, the number of check-code bits used for whatever checking scheme is employed, and the number of different words simultaneously accessed on one memory operation is to make certain odd-sounding total word lengths popular:

Organization	Total Word Length	Data Word Length	Check Bits/ Word	Checking Scheme
17x4	68	16	1	Simple parity
72x1	72	64	8	Hamming code
39x2	78	32	7	Hamming code
22x4	88	16	6	Hamming code

Table 1. Common DRAM Memory-Board Organizations

Assumptions and Equations

The key to good memory-board design is optimization of the layout and impedance of the wiring traces, and the choice of efficient RAM drivers. In prototype wirewrapped boards, the characteristic impedance of a wire which is at a varying distance from a ground plane as it crosses hill-and-dale over other wires may be difficult to control or predict, but is likely to be within the range of 100 to 120 ohms. In production memory boards, however, it is often a good approach to use **microstrips** to interconnect the array of DRAMs. A microstrip is simply a PCB wiring trace over a ground plane, separated from that ground plane by a thin layer of insulating medium such as fiberglass. A cross section of a microstrip is shown in Figure 1.

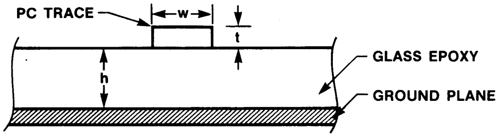


Figure 1. Microstrip Cross Section

The equations needed to design a memory board for a DRAM array interconnected by microstrips are listed below. Their rationale and derivation can be found in references on the application of electromagnetic field theory to circuit-board design[r2, r3].

Z_o = the characteristic trace impedance.

$$= \frac{87}{\sqrt{e_r + 1.41}} \ln\left(\frac{5.98h}{0.8w + t}\right) \text{ohms}$$

T_d = the trace propagation velocity.

$$= 0.0848 \sqrt{0.475e_r + 0.67} \text{ nsec/inch}$$

C_o = the trace capacitance.

$$= 1000 \,(T_d/Z_o) \text{ pf/inch}$$

C_d = the equivalent trace capacitance associated with each DRAM. It takes 0.5 inches to interconnect one DRAM.

$$= 3.5 \text{ pf}/0.5 \text{ inch} = 7 \text{ pf/inch}$$

Z'_o = the modified trace impedance due to the capacitive loading of the DRAMs.

$$= \frac{Z_o}{\sqrt{1 + C_d/C_o}}$$

T'_d = the modified trace propagation time due to the capacitive loading of the DRAMs.

$$= T_d \sqrt{1 + C_d/C_o}$$

Where:

e_r = the relative dielectric constant of the PC board.
h = the distance from the trace to the ground plane.
w = the width of the trace.
t = the thickness of the trace.

Design Approaches and Their Consequences

Very well then, let's charge right in and see what these formidable-looking equations predict will happen when a memory board is laid out in an obvious, common-sense manner. To make the example specific, we choose the 39x2 organization, so that from a circuit point of view the word length on the memory board is 78 bits. Now, each wiring trace has a capacitance (C_{TRACE}) and an inductance (L_{TRACE}) per DRAM; assuming that the DRAMs are deployed at uniform intervals along the trace, these values are determinable easily from the values per-unit-length from the microstrip equations just presented, once the spacing in inches between DRAMs has been specified. (The value for L_{TRACE} has been buried in the equation for Z_o above and won't appear in any subsequent equations.) To be specific, we'll make the realistic assumption of one DRAM per 1/2 inch of trace. Each DRAM input also has a capacitance (C_{DRAM}) and an inductance (which we're justified in neglecting); we'll assume that these are uniform, although the most sophisticated designers consider **distributions** of DRAM capacitances. The electrical situation which results is shown in Figure 2:

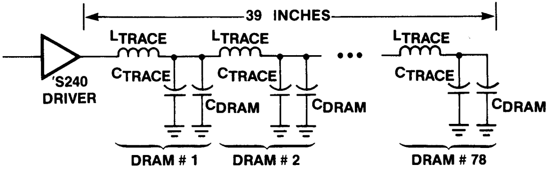


Figure 2. Transmission-Line Equivalent of a Single DRAM Wiring Trace

Typically, this trace has the following characteristics:

e_r = 5 (for G10 glass epoxy)
h = 30 mils
w = 15 mils
t = 3 mils

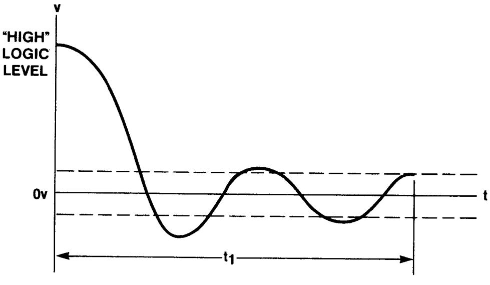
The following values can then be calculated using the appropriate equations:

Z_o = 85 ohms
T_d = 0.15 nsec/inch
C_o = 1.76 pf/inch
Z_o' = 38 ohms
T_d' = 0.35 nsec/inch

If we just string the DRAMs right down the trace like Christmas-tree lights, it will take 39 inches of trace to connect all 78 of them. So the actual propagation delay of the drive signal as it surges down this trace will be T_d' times 39 inches, or 0.35x39 = 13.7 nsec.

Notice that we are embarked on a design which is **specific** to the properties, including C_{DRAM}, of the DRAMs which we are using; a final board design is inevitably, to some extent, "tuned" to a specific DRAM type. If C_{DRAM} changes, even in what might be considered the favorable direction (smaller, obviously!), the trace impedance gets changed and the design may no longer be "tuned." But we won't worry about that here.

Now, an 'S240 driver, such as we have assumed to be driving the trace, has a signal rise time or fall time of anywhere from 2 nsec to 10 nsec, depending on semiconductor manufacturing parameters. (The rise time is, to be precise, defined as the time it takes for the output voltage to go from 10% of full-scale to 90% of full-scale; the fall time is the obvious converse.) A good rule-of-thumb for circuit-board designers is that twice the propagation delay of the trace should be less than the rise time or fall time of the driver in order to avoid serious signal **reflections,** in which a "reflected" electromagnetic wave comes bouncing back from the other end of the trace. In other words, 2x 13.7 nsec = 27.4 nsec must be less than 2-to-10 nsec, which it obviously isn't. Hence there **will** be reflections on this line, and **ringing** of the signal will occur, resulting in a waveform in the trace which looks like that of Figure 3 for a High-to-Low transition at the 'S240 output. The amplitude of the ringing voltage in real systems may be as much as 2v or even 2.5v.



t_1 = TIME TO AN ACCEPTABLE ZERO.

Figure 3. Line Ringing Due To Driver Mismatch

An 'S240 has a Schottky-driver output stage which may simplistically and approximately be represented as shown in Figure 4. When the 'S240 is driving to the logic High state, the switch S may be thought of as in position #1; when it is driving Low, S is in position #2. The effective output impedance of the 'S240 is thus about 30 ohms when driving from a previous Low state to High, but only about 10 ohms when driving from High to Low — a 3:1 difference. Thus, as the large lower transistor in the output "totem-pole" structure turns on very fast because of this low impedance, the fall time is **extremely** fast, and when ringing occurs the result may be **undershoot** — the voltage in the trace actually falls **below ground**.

An obvious consequence of ringing in the signal trace is that the system designer must allow much longer for the driver voltages, as seen by the DRAM inputs, to settle down after a transition since the ringing may be severe enough to repeatedly cross the switching threshold for the DRAMs. If this settling only had to happen once per memory access it would be bad enough, but it happens **twice** — remember that first the **row** address, and then the **column** address, gets transmitted over the address lines. Thus the allowances made for ringing cause memory performance, as measured by access time and/or cycle time, to significantly deteriorate.

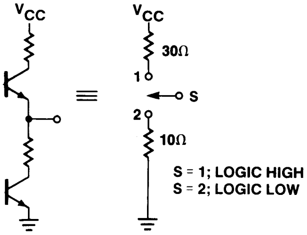


Figure 4. Typical Schottky-Driver Output Impedances

Even worse things can happen because of undershoot. First, if the voltage as seen by the DRAM inputs ever falls below -1.0v, that is, more than a volt below the steady-state PCB ground voltage at the DRAM ground pins, the contents of the "row-address registers" within the DRAMs can be altered. (**Some** DRAMs are supposed to be able to stand -2v for 20 nsec, but others just can't handle it.) Thus, if a write operation is in progress, the data word can get written helter-skelter into **different address locations in different DRAMs** (remember, each DRAM is just 1 bit wide!), so that the entire memory system very rapidly forgets everything it once knew. Second, the current surges resulting from severe undershoot may cause some 'S240-type drivers themselves to rather quickly self-destruct, which can be particularly annoying if they have been dip-soldered into place.

At this point it appears that our simple, common-sense first cut at memory-board layout is a naive recipe for disaster. So what can we do to improve on this naive approach and get the memory board to work?

First, we can **series terminate** the trace with a 10-ohm resistor to improve the impedance match. "Series termination" simply means that the resistor is located right **at** the 'S240 output, between it and the rest of the trace. 10 ohms is probably the minimum value for this resistor; other values of up to 33 ohms are also in use, according to the design context.

Second, much of our problem came about because of the sheer physical length of the trace, so we can modify the topology to cut that in half by having two "legs" rather than just one off the driver output, which should essentially cut the propagation time for the trace in half.

Third, we also if need be could vary the trace **width** w to change the trace impedance Z_0 to a value more to our liking, in order to fine-tune the design, but we won't pursue that possibility here.

The result is the significantly-different layout of Figure 5, with all of the cute little capacitors and inductors omitted for clarity (or actually for sheer laziness):

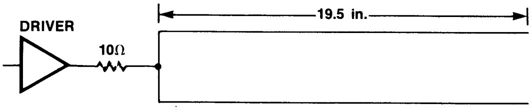


Figure 5. An Improved Layout

When the calculations are repeated, it turns out that the propagation delay down each leg of the trace is half as much, or 6.9 nsec; and the output impedances of the 'S240-plus-series-resistor are now 40 ohms when driving from Low to High, and 20 ohms when driving from High to Low, which is only a 2:1 difference. The trace impedance seen by this 'S240-plus-series-resistor is that of two 38-ohm legs in parallel, or 19 ohms, which is a very much better match to its effective output impedance. Also, the series resistor acts to slow down the exceedingly-rapid fall time of the 'S240, to the point where it may not be a great deal less than (or may even exceed) twice the trace propagation delay. So, obviously, we're a lot better off than we were.

Unfortunately, we're still not home free. We've also slowed down the **rise** time of the 'S240, i.e., the Low-to-High transition, which we weren't intending to do since it wasn't a problem. What we **really** would like is for the Low-to-High transition time and the High-to-Low transition time to become virtually the same, i.e., "symmetric." Now, DRAM addresses and data have a generally unpredictable salt-and-pepper mixture of ones and zeroes, and there is no way to take advantage under system conditions of a circuit design with one of these transition times much faster than the other. So computer-systems people, who have to be brutal realists rather than cockeyed optimists if their systems are to work reliably under real-world assumptions, normally just take whichever of these two transition times is "worse" (that is, longer) as the "logic delay" of the part as it operates within a system. Which is only reasonable! And thus it comes about that a deterioration in transition-time **symmetry** translates as a deterioration in net system **speed**.

So what do we do next? Well, we could try applying the same improvements a second time, by breaking the trace into **four** legs; however, physically interconnecting these four legs then will add more trace length, so that topology has to be traded off against interconnection efficiency. What would just get us out of this whole mess is if we could get **inside** the 'S240 and put the series resistor someplace where it will result in the effective output impedance of the driver being the same whether it is driving from Low to High or from High to Low. But we can't do **that**. Can we? Can we???

The 'S700-Family Drivers to the Rescue

Well, we can't exactly get **inside** an 'S240 and stick in a series resistor. We can, however, pull the 'S240 out of the socket it is occupying, and pop in an 'S730 — which is a **pin-compatible drop-in replacement**, and has the series resistor in exactly the right place. If we had been using a different 'S240-family driver, we could still have done the same thing — an 'S734 replaces an 'S244, an 'S700 replaces an 'S210, and an 'S731 replaces an 'S241; more on the various part types shortly.

When thus popped in as 'S240-type driver replacements, 'S700, 'S730, 'S731, and 'S734 drivers will generally speed up the total **effective** access and cycle times for most DRAM boards. This speed improvement is achieved by a sophisticated, rather than a brute-force, circuit-design approach. We've already let the cat out of the bag; they feature a new type of output stage, incorporating a **built-in** series limiting resistor, designed to efficiently drive highly-capacitative loads such as arrays of DRAM inputs interconnected by typical printed-circuit-board (PCB) wiring traces. This series resistor is located in the ideal place — between the collector of the lower output transistor in the totem-pole structure and the output pin. (See Figure 6.)

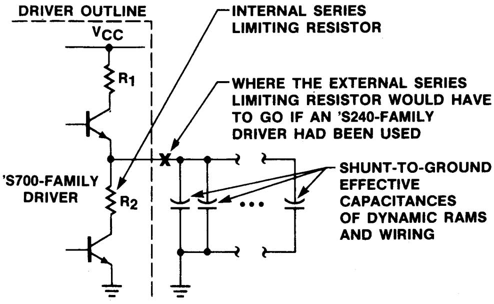

DRIVER OUTLINE
V_{CC}
R_1
'S700-FAMILY DRIVER
R_2
INTERNAL SERIES LIMITING RESISTOR
WHERE THE EXTERNAL SERIES LIMITING RESISTOR WOULD HAVE TO GO IF AN 'S240-FAMILY DRIVER HAD BEEN USED
X
SHUNT-TO-GROUND EFFECTIVE CAPACITANCES OF DYNAMIC RAMS AND WIRING

Figure 6. The Dynamic-RAM-Driver Circuit Output Stage

Now that the all-important resistor is safely inside the driver chip, its value is chosen as 20-25 ohms, so that the **in-system** Low-to-High and High-to-Low transition times of the resulting driver output stage remain symmetric, **with** the series resistor accounted for, under a wide range of circuit-loading conditions. The equivalent to Figure 4 for this new improved output stage is:

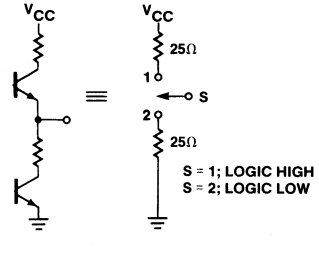
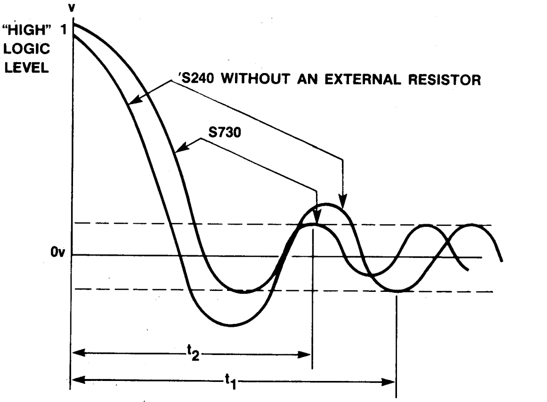


Figure 7. Driver Output Stage for 'S700-Series Buffers

What does that additional resistor in the transistor buy you? Plenty, when coupled with the other design features incorporated into the 'S700, 'S730, 'S731, and 'S734. First, there is a balanced impedance of about 25 ohms for either the Low-to-High transition or the High-to-Low transition. Since the effective impedance for the Low-to-High transition is now considerably higher than it was when using an 'S240, the undershoot problem goes away — the output voltage can never have an undershoot worse than 0.5v. Ringing **can** still occur; however, the time taken to reach an acceptable zero level is smaller than it was when using an 'S240, as shown in Figure 8.

Another advantage of the 'S700, 'S730, 'S731, and 'S734 is the high-state output voltage, now guaranteed to reach at least V_{CC}-1.15v. Certain MOS DRAM inputs are specified to require a minimum V_{IH} of 2.7 volts. More on this and other specification issues in just a minute.



t_1 = TIME TO ACCEPTABLE "LOW" LOGIC LEVEL FOR THE 'S240 WITHOUT AN EXTERNAL RESISTOR.

t_2 = TIME TO ACCEPTABLE "LOW" LOGIC LEVEL FOR THE 'S730.

Figure 8. Comparison of Undershoots; 'S240 and 'S730

Undershoot control, balanced High-state and Low-state output impedances, and appropriate voltage levels make the 'S700, 'S730, 'S731, and 'S734 very efficient RAM drivers. Consequently, although 'S240-family buffers may exhibit greater speed under light loading conditions and may even sink larger currents when operated in test jigs, 'S700-family buffers are likely to perform better under **realistic system conditions** when driving large capacitive loads is a major factor in the application. There may even be some **non-DRAM** bus-driving applications where such is the case!

And, as small added bonuses, the designer no longer has to find the physical space on his/her board for the external limiting resistors, and the resistors themselves no longer have to be paid for, and nobody has to be paid to stuff them into place on production copies of the board. All in all, an across-the-board "win-win" situation.

Keeping the Family Straight

Of the four new buffers in the 'S700 family, two — the 'S730 and 'S734 — are alternate-source versions of the Am2965 and Am2966 respectively. These two parts were originally introduced by AMD, which has also designated them alternatively as AmZ8165 and AmZ8166.

The other two buffers — the 'S700 and 'S731 — are **complementary-enable** versions of the 'S730 and 'S734 respectively, just as the 'S210 and 'S241 are complementary-enable versions of the 'S240 and 'S244. Complementary-enable parts excel in driving buses where the information to be placed on the bus can come from two different but physically adjacent origins, such as instruction addresses and data addresses in a bit-slice bipolar microcomputer system, or row-address fields and column-address fields on a DRAM memory board; more on this later.

These four new 'S700-family buffers may be grouped with Monolithic Memories' other buffers in a 2x2 matrix chart or "Karnaugh map," with the dimensions of this map chosen to be the assertiveness of the second-buffer-group enable input E_2 (here across the top, or X-axis) and the polarity of the data-buffer logical elements themselves (here down the side, or Y-axis), thus:

		Assertiveness of E_2*	
		$\overline{E_2}$	E_2
Polarity of Data Buffers	**Inverting**	'LS240 'LS340 'S240 'S340 'S730	'LS210 'LS310 'S210 'S310 'S700
	Noninverting	'LS244 'LS344 'S244 'S344 'S734	'LS241 'LS341 'S241 'S341 'S731

NOTE:

* Since $\overline{E_1}$ is assertive-low for **all** of these parts, those parts having $\overline{E_2}$ also assertive-low are "assertive-low-enable" parts, whereas those parts having E_2 assertive-high are "complementary-enable" parts.

Table 2. 8-Bit Buffers Grouped by Polarity and Enable Structure

Figure 9 shows the logic symbols for each of these four parts, in part-number order. Except for the differences already noted in the assertiveness of signal E_2, and in the output polarity of the data buffers, these parts are all mutually pin-compatible.

You will have an easier time keeping these four parts straight once you notice that the part number for one particular "architecture" of 'S700-series buffer is always the part number of the corresponding high-current buffer, **plus 490**. Since hundreds of 54/74 part numbers have already been assigned, even though not all of the corresponding parts are yet in production, obtaining part numbers with even **this** much method in the madness was not exactly a piece of cake! Anyhow, if you want to easily remember what the part number should be when you replace an 'S240-family buffer with an 'S700-family buffer, you must add 490 to its part number: e.g., 'S241 + 490 = 'S731, and so forth.

Like other Monolithic Memories 20-pin 8-bit interface circuits, the 'S700, 'S730, 'S731, and 'S734 come in the celebrated 300-mil SKINNYDIP™ package. They **also** come in eutectic-seal-flatpack and leadless-chip-carrier packages.

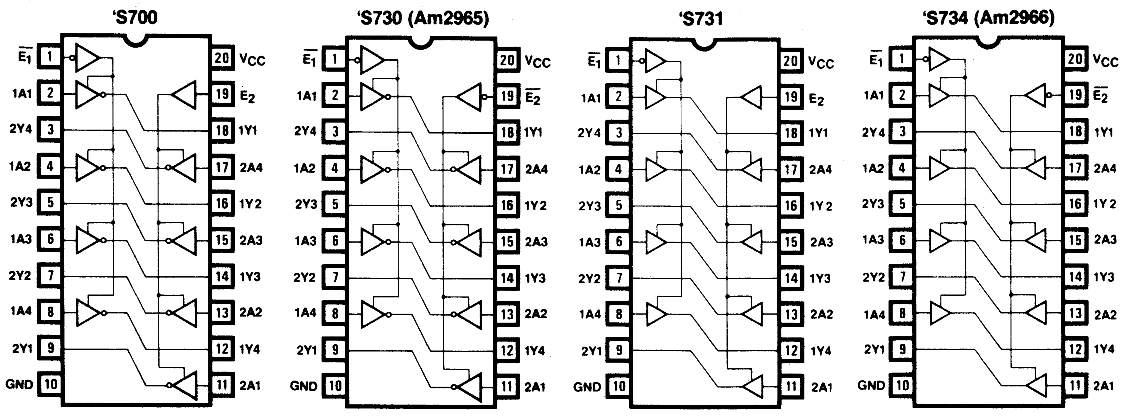
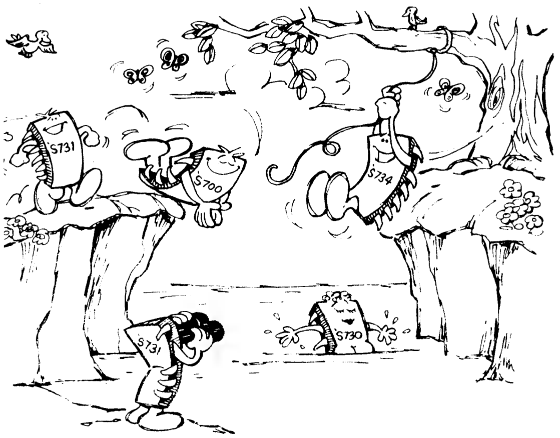


Figure 9. Logic Symbols for 'S700-Family Parts



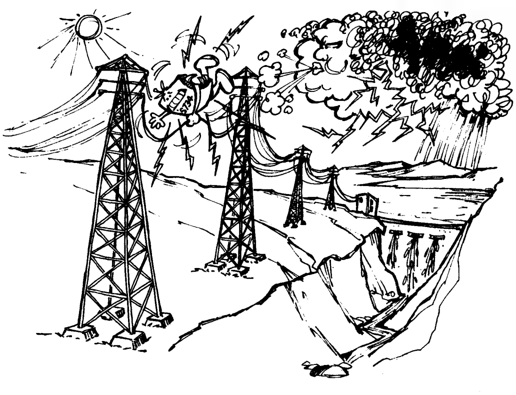
"...THE 'S700, 'S730, 'S731, AND 'S734 COME IN THE CELEBRATED 300-MIL 'SKINNYDIP™' PACKAGE..."

A Few Subtleties Regarding 'S700-Family Driver Specifications

If you are used to regular run-of-the-mill TTL data sheets, you should become sensitive to the fact that in several respects the Monolithic Memories 'S700-family data sheet (and, to be fair to a friendly competitor, AMD's Am2965/6 data sheet) represents a substantial departure from this norm.

First, since 'S700-family MOS drivers are obviously intended to mingle freely in the MOS world, they are specified to operate properly with as much as a ± 10% power-supply-level fluctuation over the entire commercial temperature range, instead of just the usual TTL ± 5%. The ± 10% standard is usual for MOS parts, but in the TTL world it is normally met only by selected military-version parts specified over the military temperature range. Thus, the V_{CC} seen by your commercial 'S700-series parts may fluctuate (even though you hope it won't) from 4.50v to 5.50v instead of only from 4.75v to 5.25v as for most commercial TTL.

Second, as already mentioned, an acceptable output logic High is considered to be V_{CC} -1.15v, or 3.85v assuming that your power supply really **is** under control after all. MOS parts are specified to think they're still seeing a Low up to 0.8v at an input, and to be seeing a High above either 2.4v or 2.7v; in between is, of course, the usual transitional or no-mans-land region. In keeping with the needs of the MOS world, 'S700-family Low-to-High logic propagation delays are measured from when the **input** crosses the usual TTL threshold somewhere in this no-mans-land (say 1.5v) to when the **output** crosses 2.7v — **not** merely to when the output crosses the TTL threshold. Likewise, 'S700-family High-to-Low logic propagation delays are measured from when the input crosses the TTL threshold to when the output crosses 0.8v. (See Figure 10 on next page.)



"...'S700 FAMILY MOS DRIVERS... ARE SPECIFIED TO OPERATE PROPERLY WITH AS MUCH AS A ±10% POWER-SUPPLY-LEVEL FLUCTUATION OVER THE ENTIRE COMMERCIAL TEMPERATURE RANGE, INSTEAD OF THE USUAL TTL ±5%..."

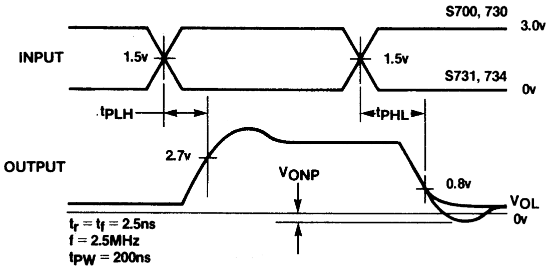


Figure 10. 'S700-Family Output-Voltage-Level Specification Conventions

Third, **both** minimum and maximum propagation delays are specified (at 25° C and 5v), so that you don't need to worry about any unwanted consequences in your system if your memory-access time for some bit positions turns out to be unexpectedly low relative to that for other bit positions. Worst-case skew between two buffer elements on the same chip is also specified.

Fourth, in keeping with the pledge that these parts can drive highly-capacitative lines, they are **specified** that way — at 500 pf loading, not only at 50 pf loading.

Fifth, unlike 'S240-family buffers, 'S700-family MOS drivers do **not** feature designed-in hysteresis.

Power-Failure-Proof Operation of Your DRAM Memory

It's generally nice if your computer, of whatever size, doesn't forget everything it was in the midst of doing and remembering if a-c power suddenly goes off. In fact, for large mainframe computers and for high-reliability control computers it may be downright critical. So, increasingly, memory designs include power-failure-protection logic, and DRAM "refresh" circuitry can run on battery-backup power. A typical design implementation is shown in Figure 11.

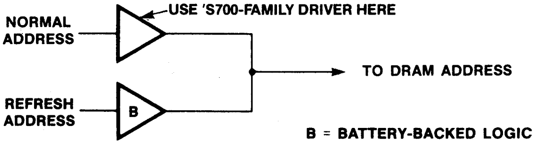


Figure 11. Battery Backup for Refresh-Address Logic

The refresh operations for the memory array must be **uninterrupted** during the transitions from a-c power to battery power and back, or else data will be lost; consequently, **all** of the logic associated with the DRAM refresh operations must be backed up. For economic reasons, other logic may **not** be backed up; hence, great care must be taken in the design at the DRAM interface, so that transients or oscillations are not introduced into the DRAM input lines by the non-backed-up logic thrashing around as a-c power goes down or comes back up.

Returning to Figure 11, note that it is the **normal** address path which is a potential source of DRAM input glitches, since the refresh-address-path buffer presumably never goes down. Again 'S700-family drivers can come riding to the rescue, since they are guaranteed to maintain glitch-free operation during either power-up or power-down.

Where to Use Complementary-Enable MOS Drivers

Driving a dynamic-MOS RAM address bus with a multiplexed row/column address can conveniently be done with an 'S700 as shown in Figure 12. This part is an inverting complementary-enable buffer with a series-resistor output structure, which is an ideal combination of characteristics here.

First of all, a TTL inverting buffer normally has one less transistor —and hence one less delay — in its internal data path than does an equivalent noninverting buffer, and hence is faster. And dynamic MOS RAMs really don't care if their addresses come in "true" or "complemented" form as long as that form **never** changes.

Second, a complementary-enable buffer can easily multiplex two different address sources to the same set of outputs without introducing extra switching delay, or allowing a momentary "bus fight" condition, if the same control signal (here CAS or "Column Address Strobe") is tied directly to both $\overline{E}_1$ and E_2, and the two 4-bit groups of outputs are tied together.

Like other three-state buffers, these parts operate in a "break-before-make" manner — it is faster to disable an output than to enable an output, by design. (The worst-case data-sheet a-c parameters don't always imply "break-before-make" operation, but the parts themselves **do** operate that way.) So, if two outputs are tied together and exchange control of the bus, they can't "fight," i.e., try simultaneously to drive the bus in opposite directions; at any given instant, one of the two will always be "floating" in the hi-Z state.

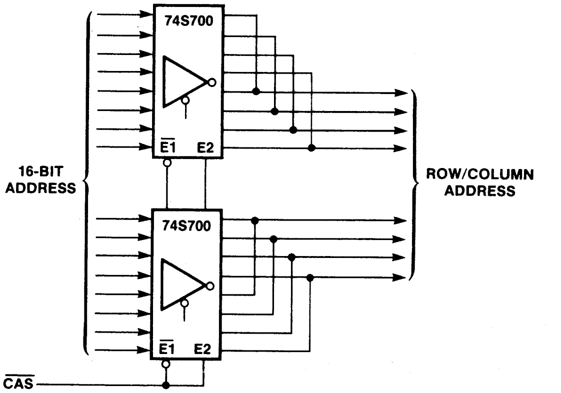


Figure 12. Multiplexed Row/Column Address Drivers

The 8 data input lines to **each** 'S700 must, of course, be parceled out with 4 lines coming from the row address and 4 lines coming from the column address.

These same advantages continue to accrue when an 'S700 is used, for example, to select between instruction addresses and data addresses in a minicomputer, or between next-micro-instruction and branch addresses in a microengine, or between input and output addresses in a multiplexed input/output data channel, assuming that in each of these cases the address being produced is to go to the DRAMs without further ado. Notice that the 'S700s here are accomplishing **driving** (that is, power amplification and impedance matching) **and multiplexing**

simultaneously. You could have used an MSI multiplexer part followed by an 'S730 to accomplish this very same thing, but with more logic delay.

If what you need in your application is a **non-inverting** driver, then everything we've just said above about the 'S700 continues to hold for the 'S731.

The Bottom Line

The 'S700, 'S730, 'S731, and 'S734, because of their unique output stage with an internal series resistor and balanced-impedance characteristics, can drive highly-capacitive loads of up to perhaps 100 dynamic-MOS RAM inputs. Since undershoot is limited to -0.5v already and so no **external** series limiting resistors are needed, the result is a net **system** speed gain, since Low-to-High and High-to-Low transition times remain symmetric. Otherwise, the logic delay would get degraded, since it must always be taken as the **worse** of these two transition times, and the use of an **external** series resistor greatly lengthens the Low-to-High transition time.

These second-generation MOS drivers also guarantee an output High voltage of V_{CC} -1.15v, and provide glitch-free operation during power-up and power-down. All of these features make them especially suitable for driving the address, data, and control lines of arrays of MOS DRAMs.

Credit Where Credit Is Due

A couple of years ago, many Monolithic Memories customers approached us with the emphatic suggestion that we should produce MOS drivers of this type, backed up by technical arguments which we have attempted herein to distil and present. In particular, the advice and assistance of Tak Watanabe of the Hewlett-Packard Computer Systems Division in Cupertino, California, has been utterly essential in the preparation of this application note.

Also, it was originally at Tak's suggestion that Monolithic Memories decided to produce the 'S700 and 'S731 complementary-enable drivers, as well as the 'S730 and 'S734 assertive-low-enable drivers. Tak's contributions, and those of other sage electronics-industry designers with whom we have spoken, are hereby gratefully acknowledged.

References

r1. *Parkinson's Law and Other Studies in Administration,* C. Northcote Parkinson, Houghton Mifflin Company, Boston, MA, 1957; also Ballantine Books, N.Y., 1964.

r2. *MECL System Design Handbook,* William R. Blood, Jr., Motorola Semiconductor Products Inc., Mesa, AZ, May 1980 (most recent edition); see in particular chapter 7.

r3. "Characteristics of Microstrip Transmission Lines," H. R. Kaupp, *IEEE Transactions on Electronic Computers,* April 1967 (Volume EC-16, Number 2); pages 185-193.

Notes

An Interface Between An SN74S409 Dynamic RAM Controller And A 68000 CPU

Frank Lee

Dynamic RAMs were introduced to increase the compactness of a memory unit in a microprocessor based system. In order to efficiently control a dynamic RAM, some controllers are needed. Most controllers are not compatible with the microprocessor used in the system, so an interface may be needed. This interface may be implemented using Programmable Array Logic devices (PALs), as in the example described in this paper (interfacing a 68000 CPU with one or more SN74S409 dynamic RAM controllers). This interface should select the desired controller, provide a refresh cycle clock to the dynamic RAM controllers and control signals to the CPU, the dynamic RAM controllers, and the dynamic RAMs. The exact implementation of other interfaces of this kind may vary depending on the CPU and the functions which can be provided by the controllers.

An Interface Between an SN74S409 Dynamic RAM Controller and a 68000 CPU

Frank Lee

Introduction

Ever since the introduction of microprocessors, compactness has been an important issue of a microcomputer. Compactness in a microcomputer involves reduction of chip counts, especially for large building blocks such as the memory. There are two basic types of memory—read only memory (ROM), and random access memory (RAM). The read only memory only constitutes a small part in the memory block, so efforts have been made to reduce the chip counts of the random access memory. There are two types of RAMS—static, and dynamic RAMs. A static RAM uses feedback circuits to constantly amplify the charges which represent the data stored in order to prevent any loss of data. The drawback is that it needs an amplifier for every data bit, which means several transistors are needed per data bit. In other words, the density of a static RAM is limited by the transistor count per bit. With the introduction of dynamic RAMs which need only one transistor per bit, the number of memory chips needed for a certain memory size has been reduced. Unlike static RAMs, a data bit in a dynamic RAM is stored in a capacitor and is easily lost due to leakage, and is therefore needed to be refreshed every certain time interval. In addition, since most dynamic RAMs are implemented as matrices with row and column addresses multiplexed, certain control signals may be needed to distinguish between row and column addresses. Therefore, a controller may be needed to do these jobs. Moreover, since different CPUs may have different control methodologies, a dedicated interface system is often needed between a CPU and a dynamic RAM controller, for example, between a 68000 and a SN74S409.

In order to see how such an interface system works, it is necessary to understand how a CPU can access a dynamic memory block.

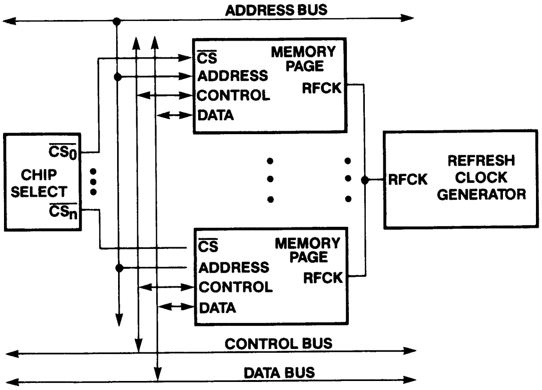


Figure 2. Inside a dynamic RAM unit.

A Typical CPU-Dynamic RAM Block

There are three basic types of signals which connect a CPU to most other units in a system—control, address, and data. The same case holds for a memory unit (see fig. 1). A memory unit may not be a simple unit by itself. It may have a number of pages (fig. 2), each of which is independently selected by a chip select signal ($\overline{CSi}$).

If a dynamic memory is used, a refresh clock (RFCK) is needed to time the refresh cycles. Normally, only one refresh clock generator is needed for all memory pages.

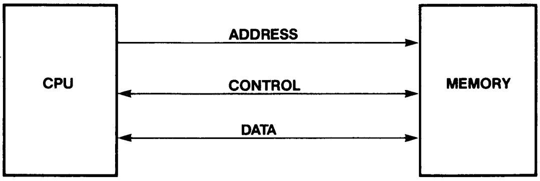


Figure 1. Communication between a CPU and a memory unit.

Inside a Memory Page

Inside a memory page, there are a number of banks (fig. 3). For a static RAM, only a simple bank select control is necessary for controlling the whole page. For a dynamic RAM, as stated in the introduction, refresh is necessary, and address bits are often multiplexed. There must be some kind of control which can provide all the necessary signals. This can be done with a dynamic RAM controller. In order for a dynamic RAM controller to talk to a CPU, an interface is often necessary. This interface will provide all the handshaking signals for the CPU and the dynamic RAM controller. In order to understand how this system works, it is necessary to know how a dynamic RAM operates. More details on dynamic RAMs will be discussed later.

It should also be noted that most dynamic RAMs on market are bit slices, so 8 chips are normally connected together to form a byte bank (fig. 4). For CPU which are word (2 bytes) addressable, two of these bytes may be tied together to form a 16-bit work bank. For some applications where 16-bit words with single-error-correction-double-error-detection are needed, a total of 22 chips are tied together to form a 22-bit Hamming-coded word storage.

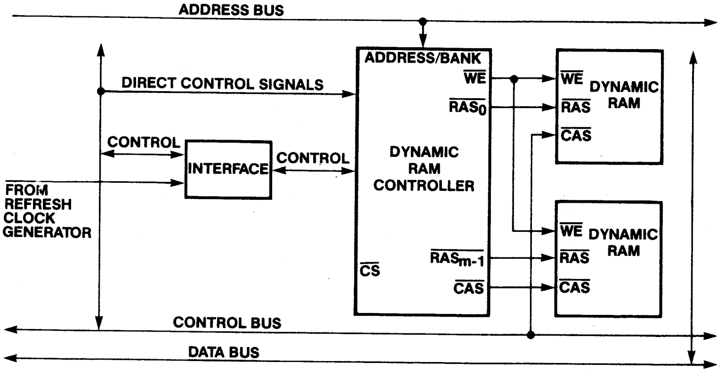


Figure 3. Inside a memory page of a dynamic memory unit.

Dynamic RAM

A very important element in the memory unit of a computer is the read-write memory, better known as the random access memory (RAM). There are two main issues which are commonly discussed about a RAM—speed and density. Ever since integrated circuits were used for memory purposes, large advances in both issues have been noted. As for a microprocessor based system, density is a more important issue than speed. There are several factors which may limit the density of a RAM chip—the power dissipation, the number of transistors per data bit, and the size of a transistor. Of these, the size of a transistor is a matter of process technology which will not be discussed in this application note. As a result of minimizing power dissipation and the number of transistors per bit, dynamic RAMs were introduced.

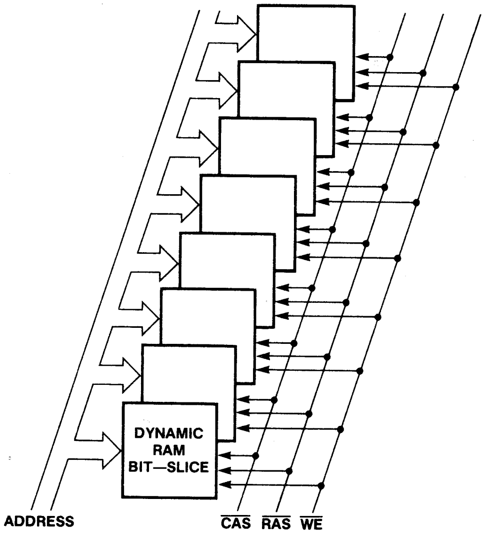


Figure 4. A graphical representation of a byte of dynamic RAMs.

Features of Dynamic RAM

The most important feature about dynamic RAM is that it requires only one transistor plus one capacitor to store a bit of data, with, of course, a certain overhead to access the memory. A data bit is stored as charge in a capacitor. For this reason, the capacitance must be quite large while the impedance of the transistor when off (i.e., the corresponding bit is not addressed) should also be very large. The most suitable circuit technology is probably MOS. In the rest of this paper, all dynamic RAM circuits used are NMOS.

Fig. 5a-c show a simple dynamic RAM which consists of a bunch of memory bits and a read-write circuit. All signals in these figures are assumed to be active HIGH. Each data bit consists of a transistor Q1 and a capacitor. For NMOS, a transistor will be on when and only when the gate is HIGH. During a memory write, write enable (WE) goes HIGH turning Q2 on while Q1 of the corresponding address is also turned on by the address. The data on the bus will be amplified and pass through Q2 and Q1 to charge up the capacitor (see fig. 5a). When Q1 and Q2 are off, the charge will remain in the capacitor.

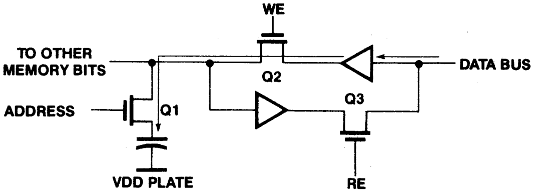


Figure 5a. WE = 1; RE = 0; address valid; data stored into capacitor.

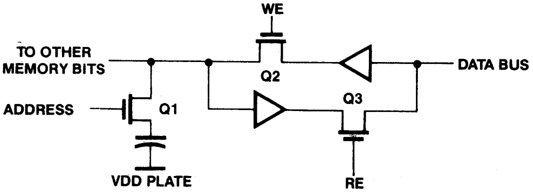


Figure 5b. WE = 0; RE = 0; or address not valid; data trapped in capacitor.

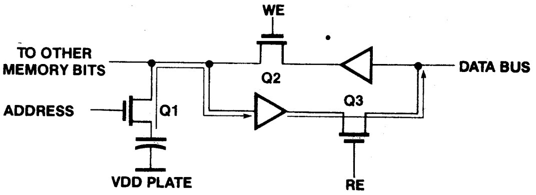


Figure 5c. WE = 0; RE = 1; address valid; data read from capacitor.

When there is a memory read, WE should be LOW, at least initially, which implies Q2 is off. Now read enable (RE) should be HIGH to turn on Q3, and the corresponding Q1 should be on, thus passing the stored data to the amplifier and then out to the bus through Q3 (see fig. 5c). Since sensing of the stored charge involves redistribution of charge to the line linking all memory bits and the sensing amplifier, it may be necessary to write the data back to the capacitor of the corresponding address in order to restore the charge. So after data is on the bus, WE should be asserted just to prevent loss of data. WE should be LOW again no later than RE goes LOW to avoid writing invalid data.

In fact, the stored charge may leak through Q1 and/or the capacitor itself even without memory read. For this reason, a read from an address (not necessary to external bus) followed by a write to the same address may be needed regularly to restore the stored charge. This process is called 'refresh'. There is no unique way to refresh a dynamic RAM. For most dynamic RAMs on market, there is another simpler way to do this job which will be discussed in the next section.

Configuration of a Dynamic RAM

With only one read-write circuit for all data bits in a dynamic RAM, it will be very likely that immediately after refreshing one bit, another bit already needs refreshing, i.e., the RAM is constantly occupied by refresh cycles. If a processor wants to access the memory, it may have to wait until a refresh cycle is over. Thus, wait states have to be asserted for nearly all memory access, which is very inefficient. Besides, every address must also be driven by some transistors which may also be very inefficient in transistor count per bit either. Worst of all, having all bits tied on a single line which links to the read-write circuitry may result in a highly capacitive line, which will lead to long delays and probable incapability to recognize data due to charge sharing between the line and data capacitors during a memory read or refresh.

Another approach is to have one read-write circuit to every bit, as in fig. 6b. Compared with the static RAM in Fig. 6a, it is easily seen that it will not have much advantage in transistor count.

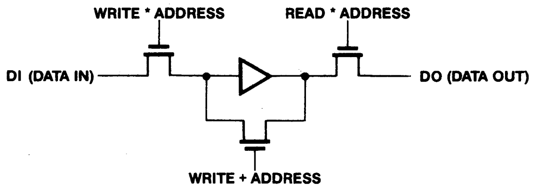


Figure 6a. A simple static RAM cell.

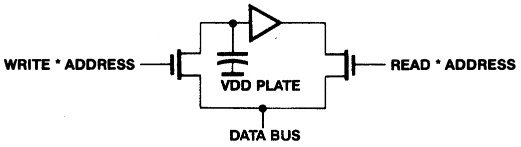


Figure 6b. A simple dynamic RAM cell with read-write circuit on it.

A more efficient way is to implement a dynamic RAM as a matrix, which is commonly used in dynamic RAM designs. There are many implementations of a matrix dynamic RAM. For a matrix implementation, the address is partitioned into two parts—row and column. Two examples are shown in fig. 7a-b. When there is a memory refresh, a refresh will be performed on a whole row of data according to the row address. For a memory access, the row address is latched in first, followed by a column address which determines the column to be accessed. It is a common practice that the row and column addresses are multiplexed and controlled by the row and column address strobes ($\overline{\text{RAS}}$ and $\overline{\text{CAS}}$), while read-write is controlled by a write enable signal ($\overline{\text{WE}}$). Note that the address strobes ($\overline{\text{RAS}}$ and $\overline{\text{CAS}}$) and the write enable ($\overline{\text{WE}}$) are normally active LOW.

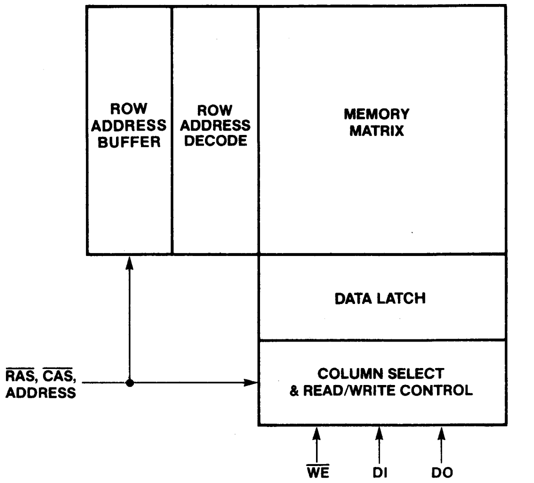

ROW ADDRESS BUFFER
ROW ADDRESS DECODE
MEMORY MATRIX
DATA LATCH
COLUMN SELECT & READ/WRITE CONTROL
$\overline{\text{RAS}}$, $\overline{\text{CAS}}$, ADDRESS
$\overline{\text{WE}}$
DI
DO

Figure 7a. An example of matrix implementation of dynamic RAM—rows and columns are not partitioned into different groups.

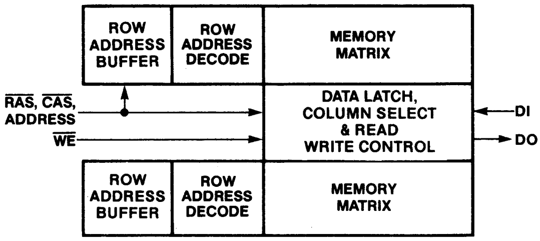


Figure 7b. An example of matrix implementation of dynamic RAM—rows partitioned into 2 groups; columns are not partitioned.

For a dynamic RAM having a matrix configuration, a refresh is done every time $\overline{\text{RAS}}$ is lowered. Details on the circuitry of dynamic RAM will not be discussed in this application note.

Limitations on Dynamic RAMs

There are certain restrictions on the input signals to a dynamic RAM:

1. Row address valid to $\overline{\text{RAS}}$. The row address setup time must be satisfied before $\overline{\text{RAS}}$ is asserted.
2. Column address valid to $\overline{\text{CAS}}$. The column address setup time must be satisfied before $\overline{\text{CAS}}$ is asserted.
3. $\overline{\text{RAS}}$ to $\overline{\text{CAS}}$ delay. The delay must be at least equal to the sum of the row address hold time and the column address setup time.

4. RAS active period. This must be at least the time in which memory refresh or access can be accomplished.
5. RAS inactive period. This must be at least the minimum precharge time for all parts of the circuits.
6. CAS active period. This should be a subinterval of RAS active period and must satisfy the RAS to CAS delay and allow memory access to be accomplished.
7. The addresses hold times, especially the row address hold time, must also be satisfied.

Some Other Information on Dynamic RAMs

Most dynamic RAMs on market today are implemented as bit slices. Some of the most popular ones are made by Hitachi, Fujitsu, and Motorola, etc. The current state-of-the-art dynamic RAMs on market are 64K by 1 bit with memory access time of approximately 100ns, although certain companies announced that they were developing 256K by 1 bit dynamic RAMs. Some companies have also developed on chip support units such as refresh counters for their dynamic RAMs.

Dynamic RAM Controller

A dynamic RAM controller is the unit which provides the necessary addresses and control signals (RAS, CAS, WE) to the dynamic RAMs. At a minimum, a dynamic RAM controller should be able to control the refresh and to multiplex the row and column addresses for the dynamic RAMs. It should also be able to provide refresh addresses although it may not be necessary if the dynamic RAMs driven have on chip refresh counter.

Features Useful Provided by a Dynamic RAM Controller

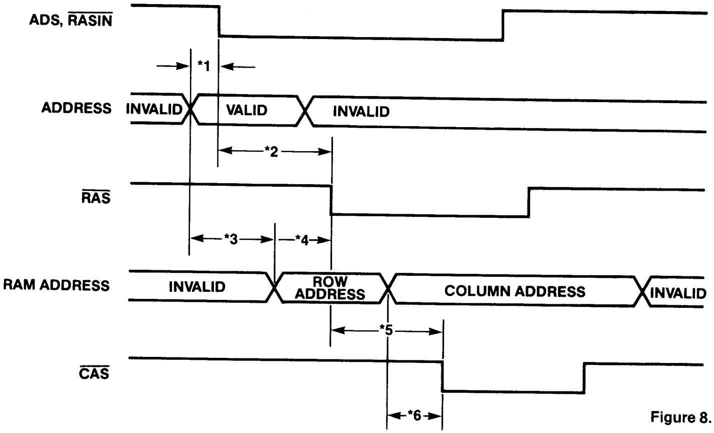
There are a lot of features which may be very useful if provided as options in a dynamic RAM controller:

1. Automatic access capability. Column address strobe (CAS) can be generated automatically after the row address strobe (RAS) when memory is accessed.
2. Refresh options:
   a) Automatic forced refresh. When a refresh is requested, the refresh address will be on the output and RAS will be automatically lowered.
   b) Automatic burst refresh. Refresh of the whole chip is done automatically, i.e., RAS will be automatically lowered for every refresh address. This is very useful before or after a DMA operation.
   c) Hidden refresh during memory access. This is to steal an unused memory cycle to perform a refresh without putting on wait states, thus maximizing efficiency.
3. Initialization options. Dynamic RAMs are very susceptible to alpha particles. For this reason, most dynamic memory systems have built-in error correction and detection capabilities. When the system is powered up, the contents of the memory is random. In order to protect the CPU from receiving error messages, the memory should be initialized to some legal contents. This can be easily done if there is an initialization option, such as an all-RAS write—writing of a legal word to all memory locations.
4. Compatibility with a CPU. Most dynamic RAM controllers are designed for general CPU-dynamic RAM system but not for any particular CPU. Therefore, for most systems, signals from or to the dynamic RAM controllers are not compatible with those of the CPU, and an interface is needed.

Electrical Characteristics of a Dynamic RAM Controller

There are other issues that are important for a dynamic RAM controller:

1. Addressing capability:
   a) Number of memory banks it can address. Note that when memory is accessed, only one bank is activated. But during a refresh, all banks are activated.
   b) Maximum size of dynamic RAM chips it can address. It depends on the number of address lines it can provide to the dynamic RAMs.
2. Driving capability. All characteristics of dynamic RAM controller are specified under certain load conditions. These conditions will be very important in determining the usefulness of the controller as well. These conditions normally suggest the limit of the number of dynamic RAMs a controller can drive directly without buffers.
3. Limitations on the speed of the dynamic RAMs it can drive. The dynamic RAM has certain speed specifications needed to be satisfied. In order to control a dynamic RAM, a dynamic RAM controller must be able to meet the following delay specifications:
   a) ADS, RASIN to RAS delay. It is assumed that ADS and RASIN are tied to a common address strobe. As seen from fig. 8, the address set up time (t_{ASA}) plus the RASIN to RAS delay (t_{RPDL}) minus the address input to output delay (t_{APD}) must be at least the minimum of the row address set up time (t_{ASR}) specified for the dynamic RAMs chosen.
   b) RAS to CAS delay for automatic access. In this mode, CAS will be asserted automatically after a certain delay (t_{RCDL}) after RAS is asserted. In this case, the minimum t_{RCDL} of the dynamic RAM controller must be at least the minimum t_{RCDL} of the dynamic RAMs controls.



*1. t_{ASA} — ADDRESS SETUP TIME LO ADS

*2. t_{RPDL} — RASIN TO RAS DELAY

*3. t_{APD} — ADDRESS INPUT TO OUTPUT DELAY

*4. t_{ASR} — ROW ADDRESS SETUP TIME (DYNAMIC RAM PARAMETER)

*5. t_{RCDL} — RAS TO CAS OUTPUT DELAY

*6. t_{ASC} — COLUMN ADDRESS SETUP TIME

Figure 8. A timing diagram for a dynamic controller (SN74S409).

Assuming no external delay circuitry is used, a dynamic RAM controller can only control those dynamic RAMs which can meet the delay requirements. If the dynamic RAMs are too slow such that the above delays are too large for the controller, then the controller may not be able to control them. There is no limit on the maximum speed of the dynamic RAMs a controller can control.

A comparison on different dynamic RAM controllers is given in table 1.

The SN74S409

In this paper, the SN74S409 is chosen as the dynamic RAM controller used in a 68000 CPU-dynamic RAM system. A block diagram of this dynamic RAM controller is attached in fig. 9. Since the 'S409 is not exactly compatible with 68000, an interface is needed. Before going on, more information on the 'S409 may be useful in designing the interface circuit.

Certain general information of the SN74S409 is provided in table 1. More information is included in the data sheet for this dynamic RAM controller. There are nine functional modes (0,1,2,3a,3b,4,5,6,7) altogether. Table 2 lists the functions of all the modes. In the design described later, only modes 1 and 5 are used. Mode 1 is automatic forced refresh, while mode 5 is automatic access with hidden refresh. Normally mode 5 is used for memory access, and a refresh clock (RFCK) is present. Each time RFCK is HIGH, a hidden refresh at a certain row address may be performed. If the same memory page (controlled by the same controller) is continuously accessed for every memory cycle during this time, the refresh which is supposed to occur cannot be performed in this memory page. When RFCK becomes LOW, it will send out a refresh request signal ($\overline{\mathrm{RFRQ}}$). An interface receives this signal and if there is no memory access to the locations addressed by the controller, it will send out a refresh signal ($\overline{\mathrm{RFSH}}$) forcing functional mode to go to 1 for a forced refresh. Else, it will wait for the end of this memory access before asserting $\overline{\mathrm{RFSH}}$. The way we can choose between mode 1 and 5 is by tying M1 and M0 of the 'S409 to LOW and HIGH respectively, and tying M2 to the $\overline{\mathrm{RFSH}}$ output of the interface.

More details will be discussed in the section on the interface.

A Brief Introduction To The 68000 CPU

The 68000 is a high performance microprocessor which can run with up to 10 MHz clock. It has 16 bits of data, thus enabling memory access of up to two bytes at a time. It has 23 bits of address (A23-A1) plus lower and upper data strobes which are very effective when determining whether a byte or a word should be addressed. The lower and upper data strobes ($\overline{\mathrm{LDS}}$ and $\overline{\mathrm{UDS}}$) are in fact the demultiplexed A0. If only one of $\overline{\mathrm{LDS}}$ or $\overline{\mathrm{UDS}}$ is asserted, the corresponding byte will be accessed. If both of them are LOW, the word indicated by the most significant 23 bits of address will be accessed. A long word may also be accessed by doing 2 word accesses.

When the memory is accessed, the CPU will assert wait states until a data transfer acknowledge ($\overline{\mathrm{DTACK}}$) signal is received. This signal is particularly useful for stretching the memory cycles during a memory refresh.

An issue which is not important for the interface but should be well noted is that the 68000 has memory-mapped I/O. So some addresses must be reserved for these purposes.

	SN74S408	SN74S409	AM2964B	INTEL8207
Addressing Capability: —max number of banks —max size of dynamic RAM	 4 64K	 4 256K	 4 64K	 4 256K
Driving Capability: —load capacitance at which electrical characteristics are measured (pF)	500	500	50	*1
Automatic Access Capability —with hidden refresh —$\overline{\mathrm{RASIN}}$ to $\overline{\mathrm{RAS}}$ delay (ns) —$\overline{\mathrm{RAS}}$ to $\overline{\mathrm{CAS}}$ delay (ns)	yes no 35 95-160	yes yes 35 95-160	no no n.a. n.a.	yes no *1 *1
Refresh options: —automatic forced refresh —automatic burst refresh	 no no	 yes yes	 no no	 yes yes
Remarks			*2	*3

*1 Data not available at the moment.
*2 Extensive external circuitry required.
*3 Compatible with certain Intel's microprocessors. But an interface is required if used in a system using most other microprocessors.

Table 1. A comparison between some dynamic RAM controllers.

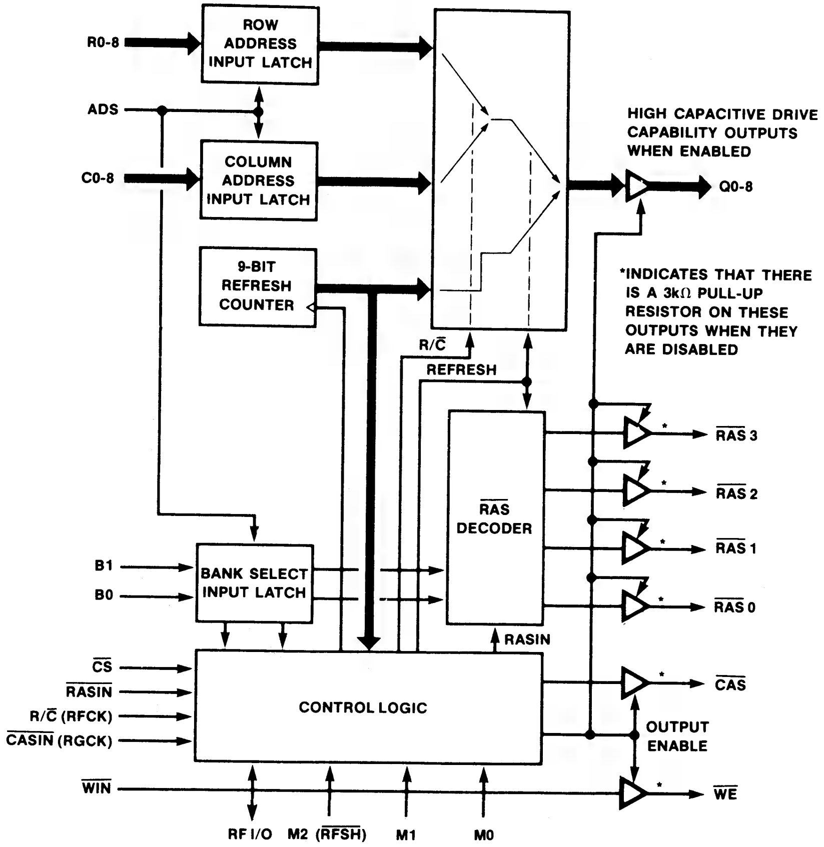


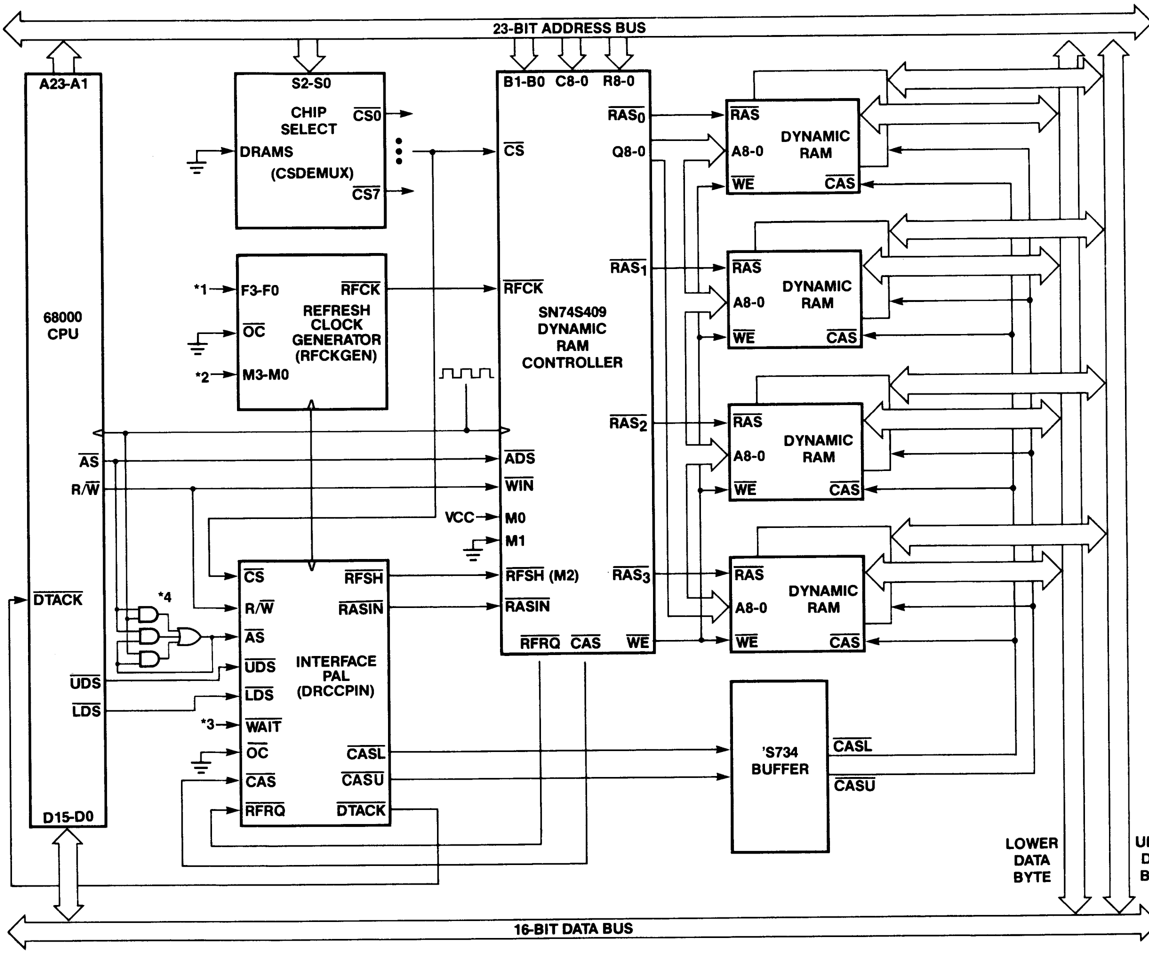
Figure 9. Block diagram of a 74S409 dynamic RAM controller.

MODE	FUNCTIONS
0	Externally controlled refresh
1	Automatic forced refresh
2	Internal automatic burst refresh
3a	All-RAS automatic write
3b	Externally controlled All-RAS access
4	Externally controlled access
5	Automatic access with hidden refresh
6	Fast automatic access
7	Set end of count

Table 2. Functions of different modes of SN74S409

Interfacing A 68000 CPU And A SN74S409 Dynamic RAM Controller

Since most dynamic RAM controllers are not compatible with the microprocessor used in a system, an interface system is often necessary between a CPU and the dynamic RAM controller(s). This kind of interface can often be implemented easily and efficiently using Programmable Array Logic chips (PALs). For example, the following describes an interface between a 68000 CPU and a 'S409 dynamic RAM controller(s). The way to implement this system using PALs for the interface circuit is shown in fig. 10. In this figure, only one page of memory is shown, but more pages may be added as desired.



*1 TIED TO VCC OR GND DEPENDING ON THE REFRESH CLOCK PERIOD CHOSEN (SEE PAL SPECIFICATIONS FOR RFCKGEN)
*2 TIED TO VCC OR GND DEPENDING ON THE DURATION OF RFCK BEING LOW (SEE PAL SPECIFICATIONS FOR RFCKGEN)
*3 TIED TO VCC IF SYSTEM IS RUNNING AT HIGHER THAN 6MHz CLOCK RATE, OR TIED TO GND IF NOT.
*4 THIS CIRCUIT IS USED TO PREVENT POTENTIAL METASTABILITY PROBLEM.

Figure 10. An interface circuit between a 68000 CPU and a 74S409 dynamic RAM controller.

To fully utilize the CPU time for such a system, it will be advantageous to use mode 5 (automatic access with hidden refresh) for the 'S409(s) as much as possible in order to minimize the chance of asserting wait states for memory refreshes. If a hidden refresh does not occur within a certain time interval, a signal RFSH will be activated and a forced refresh (mode 1) will occur. Note that when a refresh is on the way and the CPU wants to access the memory, wait states will be asserted by deactivating the data transfer acknowledge (DTACK) signal. DTACK will not be activated until one clock cycle after refresh is over. The address strobe (RASIN) will also not be asserted until one clock cycle after refresh is over. Note that since the 68000 can access a byte or a word at a time, the interface should also be able to handle this by providing signals to access the desired location(s).

Four basic functions are provided in this interface:

1. Select the desired dynamic RAM controller.
2. Provide refresh clock for the controllers. This is for doing automatic access with hidden refresh (mode 5 of 'S409).
3. Provide the handshaking signals to the CPU and the corresponding controller.
4. Provide control signals (CASL, CASU) to the dynamic RAMs. Since the controller can only provide CAS and the 68000 can only provide LDS and UDS, control signals like CASL and CASU are necessary to make sure the desired byte or word is properly addressed.

Four chips are used to provide the interface signals:

1. The dynamic RAM controller select (CSDEMUX) is a demultiplexer which can be implemented by a 32 x 8 PROM or a PAL. It is a very simple circuit and, in fact, can be implemented by a demultiplexer with minor changes. One CSDEMUX can provide chip select signals for up to eight 'S409s. This chip will not be discussed in detail, and a PAL description of it is attached in the appendix.

2. The refresh clock generator (RFCKGEN) is a programmable clock generator which can generate waveforms with various frequencies and duty cycles. Only one RFCKGEN PAL per system is necessary unless a lot of 'S409s (say, more than 16) are in the system. This PAL will be described in more detail in the next section.
3. The 68000-SN74S409 interface signals PAL (CPDRCIN) provides the handshaking signals to the CPU and dynamic RAM controller, as well as the CASL and CASU signals to the dynamic RAMs. One CPDRCIN is needed for each 'S409. Note that the outputs CASL and CASU are not sufficient to drive more than one bank of dynamic RAMs (up to approximately 22 bits). A buffer such as SN74S734 may provide these signals to up to four banks of dynamic RAMs. This PAL will also be discussed in details following the section of RFCKGEN.
4. An AND-OR gate is added to syncrhonize the AS to avoid any potential metastability problem (see fig. 10).

Descriptions for the designs (PAL specifications) of the first three chips are attached in the appendix. It should also be noted that 23-bit address of 68000 can address up to 8 pages by 4 banks by 256K by 16-bit word of dynamic RAM, i.e. 3 bits for the pages, 2 bits for the banks, 18 bits for addressing 256K dynamic RAMs through a 'S409 dynamic RAM controller. If 64K dynamic RAMs are used instead, a 68000 can address up to 32 pages by 4 banks by 64K by 16 bits.

The Refresh Clock Generator (RFCKGEN)

The refresh clock generator itself is basically a decremental counter. It has programmable operating frequency and duty cycle.

Signals:
Clock: CLK —Register clock or reference clock.
Inputs: F3-F0 —Period select of the refresh clock.
LD —Period select load. For initialization purposes.
M3-M0 —LOW duration select. Select the number of clocks cycles the RFCK output will be LOW in a period.
HOLD —Hold counter.
Output control: OC —Enable outputs when LOW.
Outputs: Q7-Q0 —Counter outputs.
INIT —Indicate end of count or LD active. Inform the counter to load in period select on the next clock.
RFCK —Refresh clock.

Operation

Initialization. When LD is LOW, INIT will be LOW the next clock. Another clock later, Q7-Q4 will be loaded with F3-F0 and Q3-Q0 with 1011 (binary).

Counting. The counter (Q7-Q0) will decrement every clock cycle until zero, which will take F3-F0•1010 (F3-F0 concatenated with 1010) cycles. During the next cycle, INIT will be LOW. One more cycle later, Q7-Q0 will be F3-F0•1011 again. Letting F3•F2•F1•F0, the number of clocks in a counter cycle will be 16f+10+2 = 16f+12.

Duty Cycle. The duration the refresh clock will be LOW is determined by the contents of M3-M0. Representing M3-M0 by m and seeing that M3-M0 are pointing to Q5-Q2, it is obvious that RFCK will be LOW for exactly 4m for each refresh clock period. Note that the duration of RFCK being LOW should not be greater than the refresh clock period, i.e., $4m \leq 16f+12$. Also, since it will be meaningless if the refresh clock RFCK is always LOW or always HIGH, 4m should not be equal to 16f+12 or 0 as well. Unfortunately, there are certain m's whose waveforms are broken into pieces, e.g., for m = 5, 9, 10, 11, and 13. These are easily noted from the equation generating RFCK. It is not recommended to use these values in real application. For all other m's, the duty cycle is defined as the duration of RFCK being HIGH in a refresh clock cycle divided by the refresh clock period, or (16f + 12 - 4m)/(16f + 12). It is recommended that the duty cycle of RFCK should be as large as possible, provided that the duration of RFCK being LOW must be long enough to process a worst case forced refresh which is described below.

The SN74S409 can perform a hidden refresh every RFCK period when RFCK is HIGH if the memory is not constantly accessed. Suppose there is a dynamic RAM with 128 refresh addresses, each of which must be refreshed every 2ms. The worst case happens when a hidden refresh occurs in a particular address immediately after RFCK becomes HIGH, and 128 refresh clock cycles later, the dynamic RAM is constantly accessed and a hidden refresh can never occur. Therefore, a forced refresh must occur when RFCK is LOW. If the CPU is still accessing that page of memory when RFCK just goes LOW, a forced refresh cannot be asserted until this memory cycle is over. If the duty cycle of RFCK is close to 1, that may mean the refresh of that particular address occurs close to 129 RFCK cycles after the last refresh. So the period of RFCK cannot be longer than 2ms/129 = 15.5us.

Suppose the system is running with a 4MHz clock. The maximum period of RFCK is 4 x 15.5 = 62 cycles. The maximum f we can choose is 3. Assuming the time needed for a worst case forced refresh to be 20 cycles, m must be at least 5. Knowing that 5 is a 'bad' number (see the paragraph on duty cycle), m should be at least 6. In order to maximize the duty cycle, f and m are chosen to be 3 and 6 respectively with duty cycle = 60%.

For different operating frequencies, different f's and may be even different m's should be chosen. Assuming m = 6 for all frequencies, a table for f and duty cycle versus frequency can be as follows:

frequency (MHz)	f(max)	duty cycle for max f(%)
4	3	60
6	5	74
8	7	81
10	9	85

Table 3. Table of Maximum f versus Frequency

The 68000-SN74S409 Interface (CPDRCIN)

This chip is to provide DTACK to the 68000, RASIN and RFSH to the SN74S409, and CASL and CASU to the dynamic RAMs controlled by that 'S409. It should be noted that if there are two or more banks of dynamic RAMs controlled by that 'S409, a driver such as SN74S734 is needed.

Signals:
clock: CLK —Register clock.
Inputs: $\overline{\text{AS}}$ —Address strobe. From 68000.
$\overline{\text{LDS}}$ —Lower data strobe. From 68000. Access lower byte.
$\overline{\text{UDS}}$ —Upper data strobe. From 68000. Access upper byte.
$\overline{\text{RW}}$ —Read/write. From 68000.
$\overline{\text{RFRQ}}$ —Refresh request. From SN74S409.
$\overline{\text{CAS}}$ —Column address strobe. From SN74S409.
$\overline{\text{CS}}$ —Chip select. From CSDEMUX.
$\overline{\text{WAIT}}$ —Wait. Tied to HIGH if 6MHz or lower, or LOW if higher.
Output Control: $\overline{\text{OC}}$ —Enable if LOW.
Outputs: $\overline{\text{CASL}}$ —Lower column address strobe. To enable lower byte.
$\overline{\text{CASU}}$ —Upper column address strobe. To enable upper byte.
$\overline{\text{RFSH}}$ —Forced refresh. To M2 of SN74S409.
$\overline{\text{RASIN}}$ —Row address strobe in. To SN74S409.
$\overline{\text{DTACK}}$ —Data transfer acknowledge. To 68000. Notify memory access is pending.

Operation:

$\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ are to be used with $\overline{\text{CAS}}$ to provide signals $\overline{\text{CASU}}$ and $\overline{\text{CASL}}$ to activate lower and upper byte(s) respectively.

$\overline{\text{RASIN}}$ is LOW whenever $\overline{\text{AS}}$ is LOW and $\overline{\text{RFSH}}$ is HIGH for at least 1 clock period. $\overline{\text{RFSH}}$ is LOW when $\overline{\text{RFRQ}}$ LOW and $\overline{\text{AS}}$ is HIGH (i.e. no memory access to the locations controlled by the corresponding 'S409) and will remain LOW for four clock cycles. $\overline{\text{DTACK}}$ is at high impedance when $\overline{\text{CS}}$ is HIGH. When $\overline{\text{CS}}$ is LOW and $\overline{\text{WAIT}}$ is HIGH, $\overline{\text{DTACK}}$ will be strobed by a memory write, an $\overline{\text{LDS}}$, an $\overline{\text{UDS}}$, or a $\overline{\text{CAS}}$ and will be HIGH an on the rising edge of $\overline{\text{LDS}}$ or $\overline{\text{UDS}}$ or both. When both $\overline{\text{CS}}$ and $\overline{\text{WAIT}}$ are LOW, $\overline{\text{DTACK}}$ will be strobed by $\overline{\text{CAS}}$ only and will again be HIGH on the rising edge of $\overline{\text{LDS}}$, $\overline{\text{UDS}}$, or both.

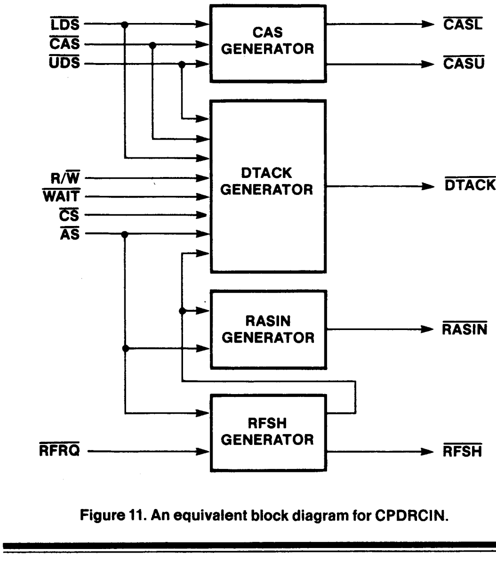


Figure 11. An equivalent block diagram for CPDRCIN.

An equivalent block diagram of this PAL is given in fig. 11.

Generation of $\overline{\text{RFSH}}$ and $\overline{\text{DTACK}}$ equations can be done effectively by using state transition diagrams (fig. 12a-b). The timing of the signals are attached in fig. 13a-b.

In order for this design to work with a 8 or 10 MHz clock, series A PALs are recommended. Note that the CSDEMUX may be replaced by some inexpensive and faster parts. The CSDEMUX is introduced here for the sole purpose of giving a simple example of what can be done if all interface chips used are PALs.

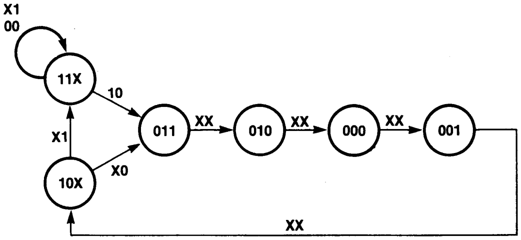


Figure 12a. State transition diagram to generate RFSH signal. States variable used are ($\overline{\text{RFSH}}$,R2,R3). R2 is used for counting the number of clock cycle $\overline{\text{RFSH}}$ being LOW as well as making sure $\overline{\text{RASIN}}$ HIGH for at least one clock cycle after $\overline{\text{RFSH}}$ changes from LOW to HIGH. R3 is used only for counting the number of clock cycles $\overline{\text{RFSH}}$ being LOW. Input variables are ($\overline{\text{AS}}$, $\overline{\text{RFRQ}}$).

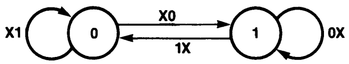


Figure 12b. State transition diagram for $\overline{\text{DTACK}}$. The state variable used is D. When D is LOW. $\overline{\text{DTACK}}$ may change from HIGH to LOW if $\overline{\text{RFSH}}$ is HIGH for at least one clock and if $\overline{\text{LDS}}$ or $\overline{\text{DDS}}$ or both is (are) LOW, or $\overline{\text{CAS}}$ is LOW, or $\overline{\text{AS}}$ and $\overline{\text{RW}}$ are LOW. When D is HIGH, $\overline{\text{DTACK}}$ may change from LOW to HIGH when $\overline{\text{LDS}}$ and $\overline{\text{UDS}}$ are HIGH. Input variables are ($\overline{\text{AS}}$, $\overline{\text{UDS}}$* $\overline{\text{LDS}}$).

Increasing Efficiency of Dynamic Memory Access

It is obvious that the most efficient way to refresh dynamic RAM is by performing hidden refresh. One way to maximize efficiency of CPU time is by maximizing the chance of hidden refresh. Seeing that if there is no memory access to the dynamic RAMs controlled by a dynamic RAM controller, a hidden refresh can be done. So it needs to minimize the chance of continuous access to the same controller all the time during the time RFCK is HIGH. It is noted that most memory accesses are either sequential or random. If memory accesses are sequential, then the memory will be accessed on continuous addresses. If these addresses belong to the same memory page (controlled by the same controller), then the controller may not be able to steal any memory cycle for refresh. One way to solve this problem is to have continuous addresses distributed over different controllers by using the least significant bits of the address to select the dynamic RAM controller. So, if there are several memory cycles during the time RFCK is HIGH, not any single dynamic RAM controller will be continuously addressed, thus enabling hidden refresh on all memory locations of the same row address. If memory accesses are random, this scheme will not be too useful, but will not be disadvantageous either. So this scheme will certainly improve the efficiency of the system.

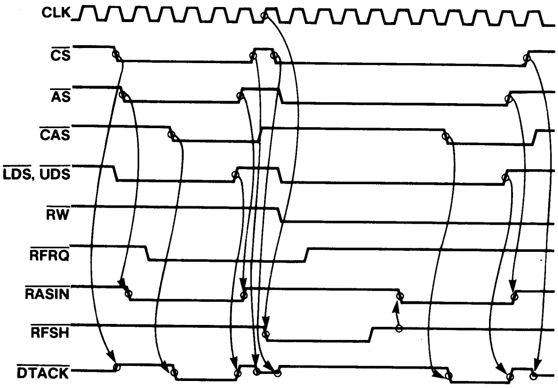


Figure 13a. Timing diagram of CPDRCIN for $\overline{WAIT}$-0. The diagram shows a read followed by a forced refresh and a write. Note that wait states can be caused by forced refresh. Moreover, it must be asserted for each memory access since the CPU is running too fast.

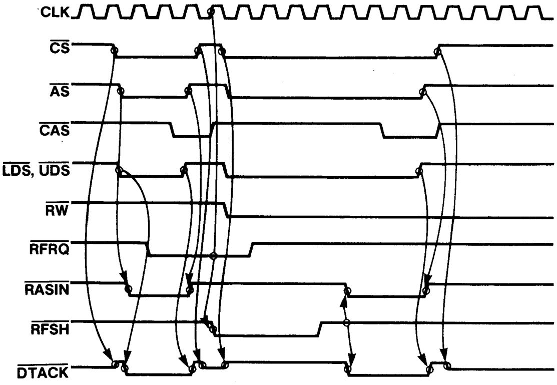


Figure 13b. Timing diagram of CPDRCIN for $\overline{WAIT}$-1. The diagram shows a read followed by a forced refresh and a write. Note that wait states can only be caused by forced refresh.

PALs For General Interface Design

For a typical CPU-dynamic RAM controller interface, PALs provide an excellent solution. Although the PAL specifications may vary with different CPUs and dynamic RAM controllers, they have the following advantages:

1) Chip count is smaller as compared to SSI logic.
2) Cost is lower as compared to dedicated logic.
3) With the extensive software and hardware supports such as PALASM and PAL programmer, design and modification of interface of this kind becomes an easy job.

Acknowledgements

I would like to thank all who helped in the preparation of this application note, especially Suneel Rajpal, Nadia Sachs, and Shlomo Waser who provided me with very important information

References

r1. **Multi-Mode Dynamic RAM Controller/Driver**—SN74-S409 data specifications, Monolithic Memories, Inc., 1165 E. Arques Avenue, Sunnyvale, CA 94086
r2. **Motorola Microprocessors Data Manual,** Motorola Semiconductor Products, Inc., pp. 4-661—4-710, 1981.
r3. **Programmable Array Logic Handbook,** Monolithic Memories, Inc., 1165 E. Arques Avenue, Sunnyvale, CA 94086.
r4. **Digital MOS Integrated Circuits,** edited by Mohamed I. ElMasry, pp. 407-454. IEEE Press, 1981.
r5. **Designer's Reference**—on semiconductor memories: PROMs and RAMs, Electronic Design, Hayden Publishing Co., Inc., 1980.

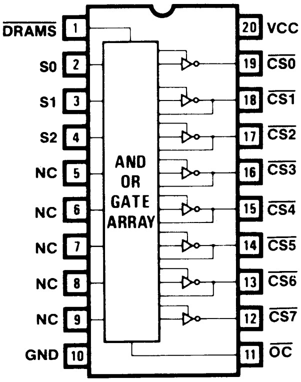
Appendix
PAL Specifications of the Interface Chips

```
PAL16L8                                            PAL DESIGN SPECIFICATION
CSDEMUX                                                  FRANK LEE 08/03/82
CHIP SELECT ADDRESS DEMULTIPLEXER
MMI SUNNYVALE, CALIFORNIA
/DRAMS S0 S1 S2 NC NC NC NC NC GND
/OC /CS7 /CS6 /CS5 /CS4 /CS3 /CS2 /CS1 /CS0 VCC


IF(OC)  CS7     =   S2* S1* S0*DRAMS      ;CONTROLLER 7 OR NO SELECT

IF(OC)  CS6     =   S2* S1*/S0*DRAMS      ;CONTROLLER 6 OR NO SELECT

IF(OC)  CS5     =   S2*/S1* S0*DRAMS      ;CONTROLLER 5 OR NO SELECT

IF(OC)  CS4     =   S2*/S1*/S0*DRAMS      ;CONTROLLER 4 OR NO SELECT

IF(OC)  CS3     =  /S2* S1* S0*DRAMS      ;CONTROLLER 3 OR NO SELECT

IF(OC)  CS2     =  /S2* S1*/S0*DRAMS      ;CONTROLLER 2 OR NO SELECT

IF(OC)  CS1     =  /S2*/S1* S0*DRAMS      ;CONTROLLER 1 OR NO SELECT

IF(OC)  CS0     =  /S2*/S1*/S0*DRAMS      ;CONTROLLER 0 OR NO SELECT


FUNCTION TABLE

/OC /DRAMS S2 S1 S0 /CS0 /CS1 /CS2 /CS3 /CS4 /CS5 /CS6 /CS7

;  /
;  D
;  R         / / / / / / / /
;/ A         C C C C C C C C
;O M  S S S  S S S S S S S S
;C S  2 1 0  0 1 2 3 4 5 6 7    COMMENTS
-----------------------------------------------------------------
 H X  X X X  Z Z Z Z Z Z Z Z    DISABLE ALL OUTPUT SIGNALS
 L H  X X X  H H H H H H H H    SET ALL OUTPUT SIGNALS HIGH
 L L  L L L  L H H H H H H H    SELECT CONTROLLER 0
 L L  L L H  H L H H H H H H    SELECT CONTROLLER 1
 L L  L H L  H H L H H H H H    SELECT CONTROLLER 2
 L L  L H H  H H H L H H H H    SELECT CONTROLLER 3
 L L  H L L  H H H H L H H H    SELECT CONTROLLER 4
 L L  H L H  H H H H H L H H    SELECT CONTROLLER 5
 L L  H H L  H H H H H H L H    SELECT CONTROLLER 6
 L L  H H H  H H H H H H H L    SELECT CONTROLLER 7
-----------------------------------------------------------------
```

DESCRIPTION

THIS PAL IS THE ADDRESS DECODER FOR THE DRAM CONTROLLER. IT CAN DECODE THREE BITS OF ADDRESS AND DRIVE EIGHT DRAM CONTROLLERS. IT CAN ALSO SET ALL OUTPUTS TO HIGH OR TO HIGH IMPEDANCE IF DESIRED.

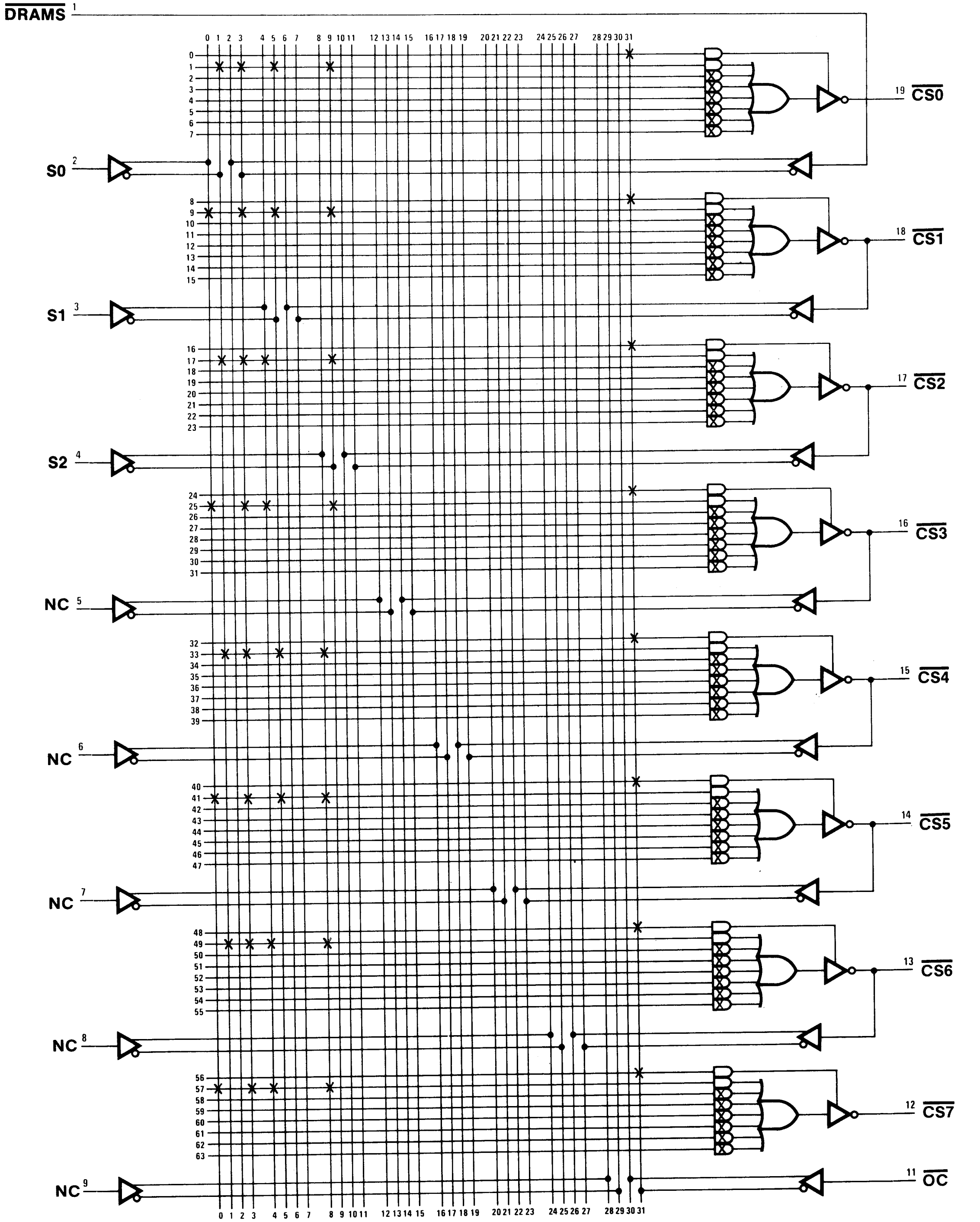
PAL16L8



```
CHIP SELECT ADDRESS DEMULTIPLEXER

                 11 1111 1111 2222 2222 2233
   0123 4567 8901 2345 6789 0123 4567 8901

 0 ---- ---- ---- ---- ---- ---- ---- ---X OC
 1 -X-X -X-- -X-- ---- ---- ---- ---- ---- /S2*/S1*/S0*DRAMS

 8 ---- ---- ---- ---- ---- ---- ---- ---X OC
 9 X--X -X-- -X-- ---- ---- ---- ---- ---- /S2*/S1*S0*DRAMS

16 ---- ---- ---- ---- ---- ---- ---- ---X OC
17 -X-X X--- -X-- ---- ---- ---- ---- ---- /S2*S1*/S0*DRAMS

24 ---- ---- ---- ---- ---- ---- ---- ---X OC
25 X--X X--- -X-- ---- ---- ---- ---- ---- /S2*S1*S0*DRAMS

32 ---- ---- ---- ---- ---- ---- ---- ---X OC
33 -X-X -X-- X--- ---- ---- ---- ---- ---- S2*/S1*/S0*DRAMS

40 ---- ---- ---- ---- ---- ---- ---- ---X OC
41 X--X -X-- X--- ---- ---- ---- ---- ---- S2*/S1*S0*DRAMS

48 ---- ---- ---- ---- ---- ---- ---- ---X OC
49 -X-X X--- X--- ---- ---- ---- ---- ---- S2*S1*/S0*DRAMS

56 ---- ---- ---- ---- ---- ---- ---- ---X OC
57 X--X X--- X--- ---- ---- ---- ---- ---- S2*S1*S0*DRAMS

LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)

NUMBER OF FUSES BLOWN =  472
```

Address Select Demultiplexer

Logic Diagram PAL16L8



```
PAL20X10                                               PAL DESIGN SPECIFICATION
RFCKGEN                                                      FRANK LEE 08/03/82
REFRESH CLOCK GENERATOR
MMI SUNNYVALE, CALIFORNIA
CLK /LD F3 F2 F1 F0 M3 M2 M1 M0 /HOLD GND
/OC RFCK Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 /INIT VCC


INIT    :=  /Q7*/Q6*/Q5*/Q4*/Q3*/Q2*/Q1*/Q0          ;END OF COUNT
         +  LD                                       ;LOAD

/Q0     :=  /INIT*/Q0                                ;DECREMENT
        :+: /INIT*/HOLD                              ;HOLD

/Q1     :=  /INIT*/Q1                                ;DECREMENT
        :+: /INIT*/HOLD*/Q0                          ;HOLD

/Q2     :=   INIT                                    ;LOAD IN 0
         +  /INIT*/Q2                                ;DECREMENT
        :+: /INIT*/HOLD*/Q0*/Q1                      ;HOLD

/Q3     :=  /INIT*/Q3                                ;DECREMENT
        :+: /INIT*/HOLD*/Q0*/Q1*/Q2                  ;HOLD

/Q4     :=   INIT*/F0                                ;LOAD IN F0
         +  /INIT*/Q4                                ;DECREMENT
        :+: /INIT*/HOLD*/Q0*/Q1*/Q2*/Q3              ;HOLD

/Q5     :=   INIT*/F1                                ;LOAD IN F1
         +  /INIT*/Q5                                ;DECREMENT
        :+: /INIT*/HOLD*/Q0*/Q1*/Q2*/Q3*/Q4          ;HOLD

/Q6     :=   INIT*/F2                                ;LOAD IN F2
         +  /INIT*/Q6                                ;DECREMENT
        :+: /INIT*/HOLD*/Q0*/Q1*/Q2*/Q3*/Q4*/Q5      ;HOLD

/Q7     :=   INIT*/F3                                ;LOAD IN F3
         +  /INIT*/Q7                                ;DECREMENT
        :+: /INIT*/HOLD*/Q0*/Q1*/Q2*/Q3*/Q4*/Q5*/Q6  ;HOLD

/RFCK   :=  M3*/Q7*/Q6* Q5                           ;32 LOW STATES
         +  M2*/Q7*/Q6*/Q5* Q4                       ;16 LOW STATES
        :+: M1*/Q7*/Q6*/Q5*/Q4* Q3                   ;8 LOW STATES
         +  M0*/Q7*/Q6*/Q5*/Q4*/Q3* Q2               ;4 LOW STATES
```

```
FUNCTION TABLE

CLK /OC /LD /HOLD F3 F2 F1 F0 M3 M2 M1 M0
/INIT Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 RFCK

;      /             /
;      H             I          R
;C / / O             N          F
;L O L L FFFF MMMM   I QQQQQQQQ C
;K C D D 3210 3210   T 76543210 K    COMMEMTS
---------------------------------------------------------------------------
 C L L X LLHH LLHH   L XXXXXXXX X    READY TO LOAD COUNTER
 C L L H LLHH LLHH   L LLHHHLHH X    READY TO LOAD COUNTER
 C L H H LLHH LLHH   H LLHHHLHH H    COUNTER LOADED, READY TO COUNT DOWN
 C L H H LLHH LLHH   H LLHHHLHL H    COUNT DOWN
 C L H H LLHH LLHH   H LLHHHLLH H    COUNT DOWN
 C L L H LLLH LHHL   L LLHHHLLL H    READY TO LOAD COUNTER, COUNT DOWN
 C L H H LLLH LHHL   H LLLHHLHH H    COUNTER LOADED
 C L H H LLLH LHHL   H LLLHHLHL L    COUNT DOWN
 C L H H LLLH LHHL   H LLLHHLLH L    COUNT DOWN
 C L H H LLLH LHHL   H LLLHHLLL L    COUNT DOWN
 C L L H LLLH LHHL   L LLLHLHHH L    READY TO LOAD COUNTER, COUNT DOWN
 C L H H LLLL LHHL   H LLLLHLHH L    COUNTER LOADED
 C L H H LLLL LHHL   H LLLLHLHL L    COUNT DOWN
 C L H H LLLL LHHL   H LLLLHLLH L    COUNT DOWN
 C L H H LLLL LHHL   H LLLLHLLL L    COUNT DOWN
 C L H H LLLL LHHL   H LLLLLHHH L    COUNT DOWN
 C L H H LLLL LHHL   H LLLLLHHL H    COUNT DOWN, RFCK HIGH
 C L H H LLLL LHHL   H LLLLLHLH H    COUNT DOWN, RFCK HIGH
 C L H L LLLL LHHL   H LLLLLHLH H    HOLD
 C L H H LLLL LHHL   H LLLLLHLL H    COUNT DOWN
 C L H H LLLL LHHL   H LLLLLLHH H    COUNT DOWN
 C L H H LLLL LHHL   H LLLLLLHL H    COUNT DOWN
 C L H H LLLL LHHL   H LLLLLLLH H    COUNT DOWN
 C L H H LLLL LHHL   H LLLLLLLL H    COUNT DOWN
 C L H H LLLL LHHL   L HHHHHHHH H    COUNT DOWN, READY TO LOAD COUNTER
 C H X X XXXX XXXX   Z ZZZZZZZZ Z    HIGH IMPEDANCE TEST
---------------------------------------------------------------------------
```

DESCRIPTION

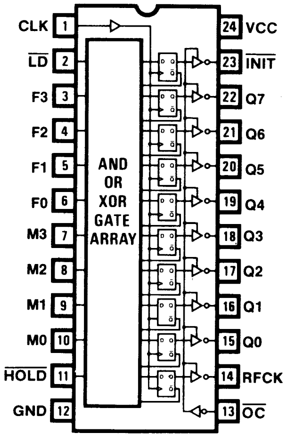
THE REFRESH CLOCK GENERATOR CAN GENERATE REFRESH CLOCK (RFCK) FOR THE DYNAMIC RAM CONTROLLERS. THE PERIOD OF RFCK DEPENDS ON F3-F0 WHILE THE DURATION OF RFCK BEING LOW DEPENDS ON M3-M0.

FFFF 3210	RFCK PERIOD (CYCLES)
0000	12
0001	28
0010	44
0011	60
0100	76
0101	92
0110	108
0111	124
1000	140
1001	156
1010	172
1011	188
1100	204
1101	220
1110	236
1111	252

MMMM 3210	RFCK LOW DURATION (CYCLES)
0000	0
0001	4
0010	8
0011	12
0100	16
0101	20*
0110	24
0111	28
1000	32
1001	36*
1010	40*
1011	44*
1100	48
1101	52*
1110	56
1111	60

*NOT ALLOWED DUE TO BAD WAVEFORMS

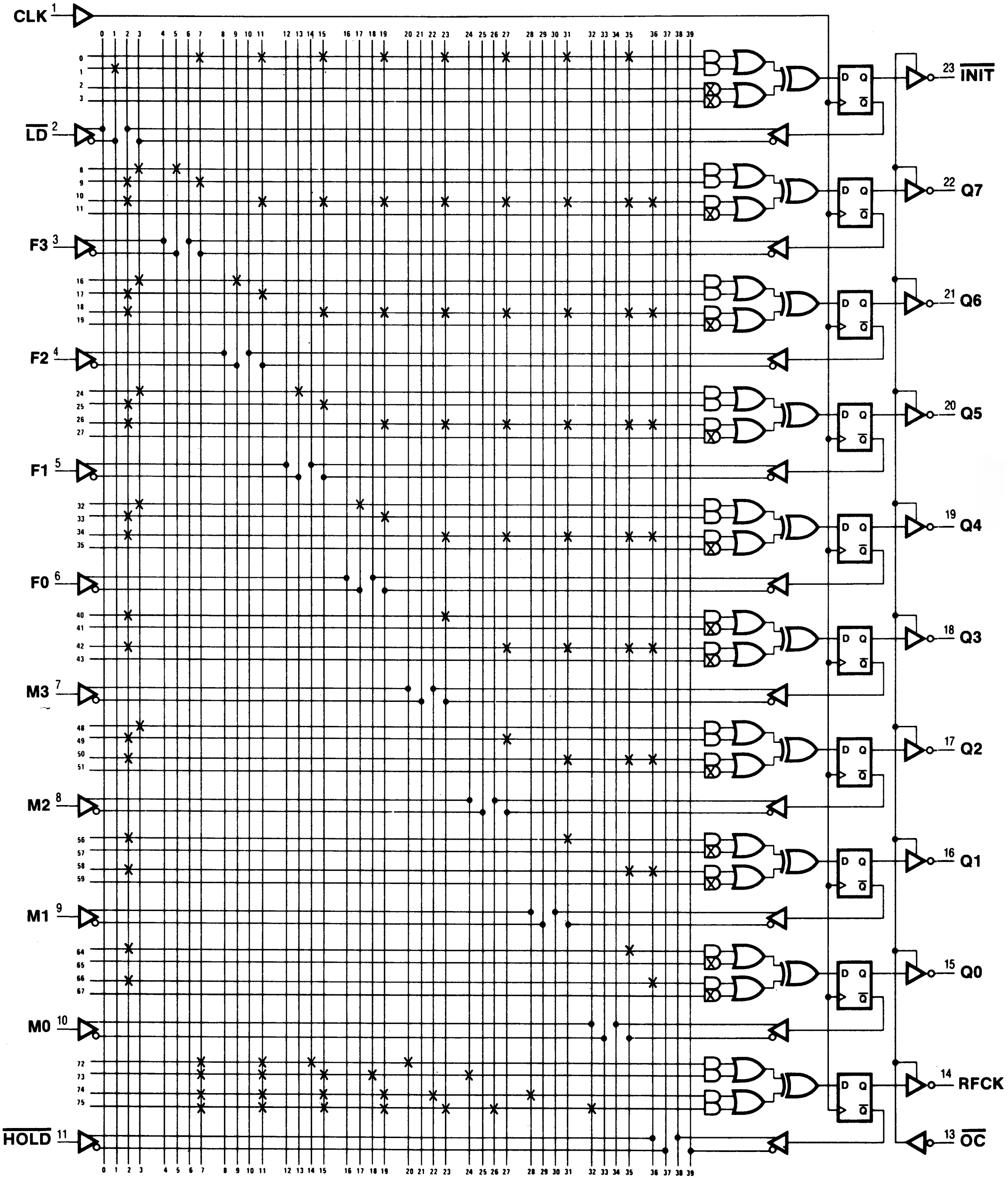
PAL20X10



```
REFRESH CLOCK GENERATOR

                 11 1111 1111 2222 2222 2233 3333 3333
   0123 4567 8901 2345 6789 0123 4567 8901 2345 6789

 0 ---- ---X ---X ---X ---X ---X ---X ---X ---X ---- /Q7*/Q6*/Q5*/Q4*/Q3*/Q2-
 1 -X-- ---- ---- ---- ---- ---- ---- ---- ---- ---- LD

 8 ---X -X-- ---- ---- ---- ---- ---- ---- ---- ---- INIT*/F3
 9 --X- ---X ---- ---- ---- ---- ---- ---- ---- ---- /INIT*/Q7
10 --X- ---- ---X ---X ---X ---X ---X ---X ---X X--- /INIT*/HOLD*/Q0*/Q1*/Q2-

16 ---X ---- -X-- ---- ---- ---- ---- ---- ---- ---- INIT*/F2
17 --X- ---- ---X ---- ---- ---- ---- ---- ---- ---- /INIT*/Q6
18 --X- ---- ---- ---X ---X ---X ---X ---X ---X X--- /INIT*/HOLD*/Q0*/Q1*/Q2-

24 ---X ---- ---- -X-- ---- ---- ---- ---- ---- ---- INIT*/F1
25 --X- ---- ---- ---X ---- ---- ---- ---- ---- ---- /INIT*/Q5
26 --X- ---- ---- ---- ---X ---X ---X ---X ---X X--- /INIT*/HOLD*/Q0*/Q1*/Q2-

32 ---X ---- ---- ---- -X-- ---- ---- ---- ---- ---- INIT*/F0
33 --X- ---- ---- ---- ---X ---- ---- ---- ---- ---- /INIT*/Q4
34 --X- ---- ---- ---- ---- ---X ---X ---X ---X X--- /INIT*/HOLD*/Q0*/Q1*/Q2-

40 --X- ---- ---- ---- ---- ---X ---- ---- ---- ---- /INIT*/Q3
42 --X- ---- ---- ---- ---- ---- ---X ---X ---X X--- /INIT*/HOLD*/Q0*/Q1*/Q2

48 ---X ---- ---- ---- ---- ---- ---- ---- ---- ---- INIT
49 --X- ---- ---- ---- ---- ---- ---X ---- ---- ---- /INIT*/Q2
50 --X- ---- ---- ---- ---- ---- ---- ---X ---X X--- /INIT*/HOLD*/Q0*/Q1

56 --X- ---- ---- ---- ---- ---- ---- ---X ---- ---- /INIT*/Q1
58 --X- ---- ---- ---- ---- ---- ---- ---- ---X X--- /INIT*/HOLD*/Q0

64 --X- ---- ---- ---- ---- ---- ---- ---- ---X ---- /INIT*/Q0
66 --X- ---- ---- ---- ---- ---- ---- ---- ---- X--- /INIT*/HOLD

72 ---- ---X ---X --X- ---- X--- ---- ---- ---- ---- M3*/Q7*/Q6*Q5
73 ---- ---X ---X ---X --X- ---- X--- ---- ---- ---- M2*/Q7*/Q6*/Q5*Q4
74 ---- ---X ---X ---X ---X --X- ---- X--- ---- ---- M1*/Q7*/Q6*/Q5*/Q4*Q3
75 ---- ---X ---X ---X ---X ---X --X- ---- X--- ---- M0*/Q7*/Q6*/Q5*/Q4*/Q3*-


LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)

NUMBER OF FUSES BLOW =  980
```

Refresh Clock Generator

Logic Diagram PAL20X10

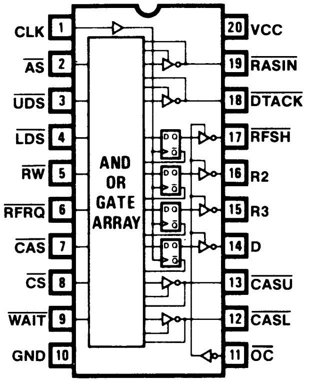


```
PAL16R4                                          PAL DESIGN SPECIFICATION
DRCCPIN                                               FRANK LEE 08/03/82
74S409 DRAM CONTROLLER - 68000 CPU INTERFACE
MMI SUNNYVALE, CALIFORNIA
CLK /AS /UDS /LDS /RW /RFRQ /CAS /CS /WAIT GND
/OC /CASL /CASU D R3 R2 /RFSH /DTACK /RASIN VCC


IF(VCC) CASL  =  CAS*LDS                         ;LOWER COL ADDR STROBE

IF(VCC) CASU  =  CAS*UDS                         ;UPPER COL ADDR STROBE

RFSH    :=  RFRQ*/RFSH*/R2                       ;/RFSH FROM STATE EQTS.
         +  RFRQ*/AS*/RFSH
         +  RFSH*R2
         +  RFSH*/R3

/R2     :=  RFSH*/R2                             ;STATE VARIABLE FOR /RFSH
         +  RFSH*/R3

/R3     :=  RFSH*R2                              ;STATE VARIABLE FOR /RFSH

/D      :=  /UDS*/LDS*/D                         ;/LDS AND /UDS HIGH
         +  /AS*D                                ;/AS HIGH

IF(CS)  DTACK =  UDS*/WAIT*/D*AS*/RFSH*R2        ;/UDS LOWERED
              +  LDS*/WAIT*/D*AS*/RFSH*R2        ;/LDS LOWERED
              +  AS*RW*/WAIT*/D*/RFSH*R2         ;WRITE
              +  CAS*/D*AS*/RFSH*R2              ;/CAS LOWERED
              +  UDS*D*AS*/RFSH*R2               ;WAIT UNTIL /UDS HIGH
              +  LDS*D*AS*/RFSH*R2               ;   AND /LDS HIGH

IF(VCC) RASIN =  AS*/RFSH*R2                     ;/RASIN SIGNAL
```

```
FUNCTION TABLE

CLK /OC /AS /UDS /LDS /RW /RFRQ /CAS /CS /WAIT
/CASL /CASU /RFSH R2 R3 /DTACK D /RASIN

;                              /  /
;            /     /   / / /   D  R
;      / /   R /   W   C C R   T  A
;C / / U L / F C / A   A A F   A  S
;L O A D D R R A C I   S S SRR C  I
;K C S S S W Q S S T   L U H23 KD N    COMMENTS
-----------------------------------------------------------------------------
 C L H H H H H H H H   H H LLH ZL H    INITIALIZE
 C L H H H H H H H H   H H HLX ZL H    INITIALIZE
 C L H H H H H H H H   H H HXX ZL H    STILL INITIALIZING
 C L H H H H H H H H   H H HHX ZL H    INITIALIZED
 C L L L L L L H L H   H H HHX LH L    WRITE
 C L L L H L L L L H   H L HHX LH L    WRITE UPPER BYTE
 C L L H L L L L L H   L H HHX LH L    WRITE LOWER BYTE
 C L L H H H L H L H   H H HHX HH L    DATA TRANSFER END
 C L H H H H L H L H   H H LHH HL H    REFRESH
 C L L L L L H H L H   H H LHL HH H    /RASIN INHIBITED
 C L L L L L H H L H   H H LLL HH H    WAIT FOR END OF REFRESH
 C L L L L L H H L H   H H LLH HH H    WAIT FOR END OF REFRESH
 C L L L L L H H L H   H H HLX HH H    WAIT FOR ONE MORE CYCLE
 C L L L L L H H L H   H H HHX LH L    /RASIN & /DTACK ASSERTED
 C L H H H H H H L H   H H HHX HL H    ALL OUTPUT SIGNALS NOT ACTIVE
 C L L L H H H H L H   H H HHX LH L    /RASIN & /DTACK BY /AS & /UDS
 C L H H H H H H L H   H H HHX HL H    ALL HIGHS
 C L L H H H H L L H   H H HHX LL L    /RASIN & /DTACK BY /AS & /CAS
 C L L L H H H L L H   H L HHX LH L    /UDS
 C L L H H H H L L H   H H HHX HH L    END OF /UDS
 C L H H H H H H L H   H H HHX HL H    END OF MEMORY CYCLE
 C H X X X X X X X X   X X ZZZ XZ X    HIGH IMPEDANCE TEST
-----------------------------------------------------------------------------
DESCRIPTION

THIS INTERFACE PROVIDES 5 INTERFACE SIGNALS FOR A 68000 CPU AND A 74S409
DYNAMIC RAM CONTROLLER, AND A NUMBER OF DYNAMIC RAMS:
     1) /DTACK -- DATA TRANSFER ACKNOWLEDGED TO THE 68000;
     2) /RASIN -- ROW ADDRESS STROBE INPUT TO THE 74S409;
     3) /RFSH  -- FORCED REFRESH SIGNAL TO THE 74S409;
     4) /CASL  -- LOWER COLUMN ADDRESS STROBE TO DRAM BANKS.
     5) /CASU  -- UPPER COLUMN ADDRESS STROBE TO DRAM BANKS.

PLEASE NOTE THAT THIS INTERFACE PAL IS A SERIES 'A' PAL (PAL16R4A).
```

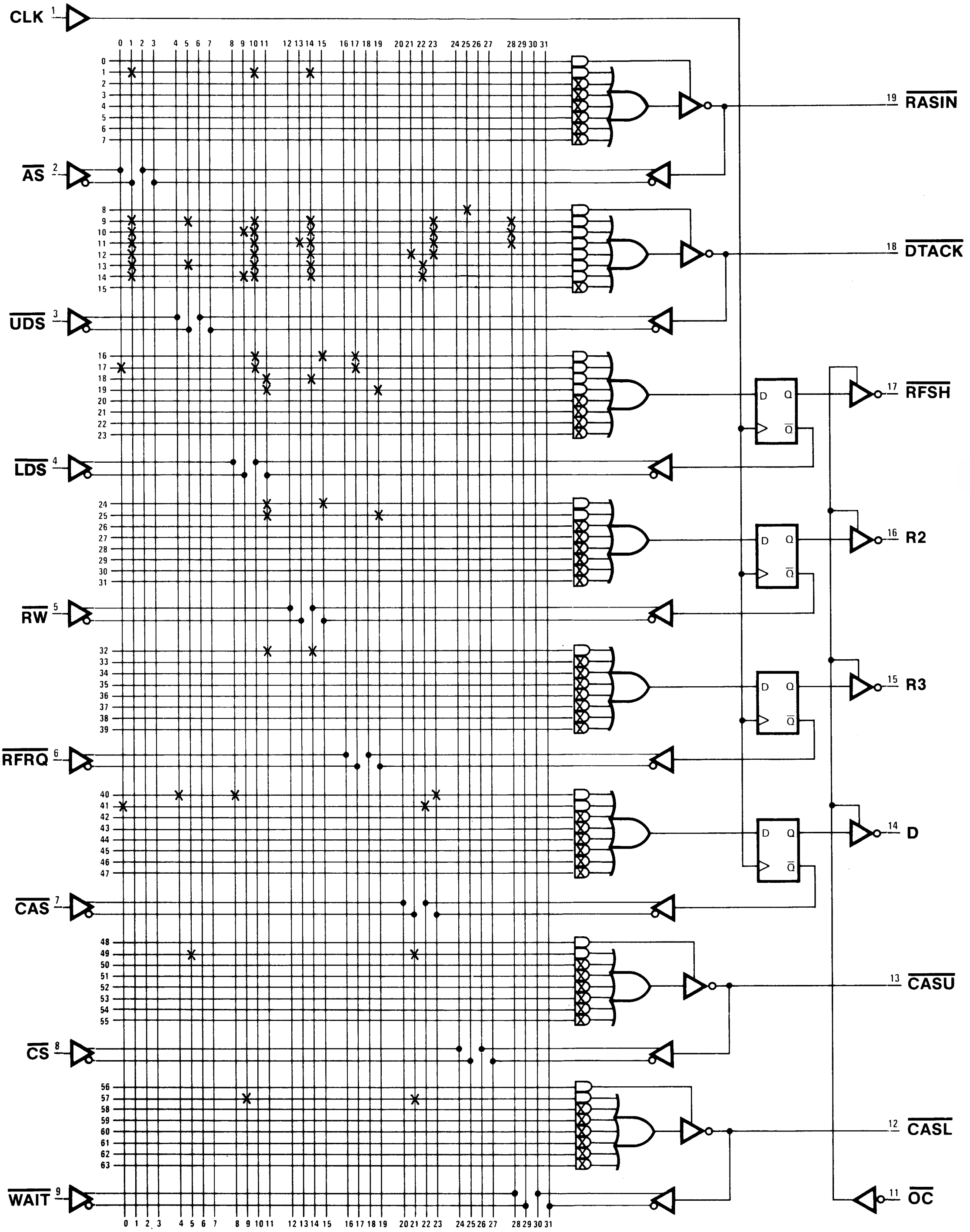
```
74S409 DRAM CONTROLLER - 68000 CPU INTERFACE

                11 1111 1111 2222 2222 2233
   0123 4567 8901 2345 6789 0123 4567 8901

 0 ---- ---- ---- ---- ---- ---- ---- ----
 1 -X-- ---- --X- --X- ---- ---- ---- ---- AS*/RFSH*R2

 8 ---- ---- ---- ---- ---- ---- -X-- ---- CS
 9 -X-- -X-- --X- --X- ---- ---X ---- X--- UDS*/WAIT*/D*AS*/RFSH*R2
10 -X-- ---- -XX- --X- ---- ---X ---- X--- LDS*/WAIT*/D*AS*/RFSH*R2
11 -X-- ---- --X- -XX- ---- ---X ---- X--- AS*RW*/WAIT*/D*/RFSH*R2
12 -X-- ---- --X- --X- ---- -X-X ---- ---- CAS*/D*AS*/RFSH*R2
13 -X-- -X-- --X- --X- ---- --X- ---- ---- UDS*D*AS*/RFSH*R2
14 -X-- ---- -XX- --X- ---- --X- ---- ---- LDS*D*AS*/RFSH*R2

16 ---- ---- --X- ---X -X-- ---- ---- ---- RFRQ*/RFSH*/R2
17 X--- ---- --X- ---- -X-- ---- ---- ---- RFRQ*/AS*/RFSH
18 ---- ---- ---X --X- ---- ---- ---- ---- RFSH*R2
19 ---- ---- ---X ---- ---X ---- ---- ---- RFSH*/R3

24 ---- ---- ---X ---X ---- ---- ---- ---- RFSH*/R2
25 ---- ---- ---X ---- ---X ---- ---- ---- RFSH*/R3

32 ---- ---- ---X --X- ---- ---- ---- ---- RFSH*R2

40 ---- X--- X--- ---- ---- ---X ---- ---- /UDS*/LDS*/D
41 X--- ---- ---- ---- ---- --X- ---- ---- /AS*D

48 ---- ---- ---- ---- ---- ---- ---- ----
49 ---- -X-- ---- ---- ---- -X-- ---- ---- CAS*UDS

56 ---- ---- ---- ---- ---- ---- ---- ----
57 ---- ---- -X-- ---- ---- -X-- ---- ---- CAS*LDS

LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)

NUMBER OF FUSES BLOWN =  642
```

PAL16R4



74S409 DRAM Controller — 68000 CPU Interface

Logic Diagram PAL16R4



Notes

FIFOs: Rubber-Band Memories to Hold Your System Together

Chuck Hastings

Data-rate matching problems are a very basic part of the life of a builder of digital systems. Today there are components called "FIFOs" which let you keep your hardware design simple, and let each portion of your system see the data rate which it wants to see, and yet let you avoid hobbling the performance of your software by constantly interrupting or intermittently healting your microprocessor. FIFO is one of those made-up words, or *acronyms,* formed from the initials of a phrase—in this case, "*F*irst-*I*n, *F*irst-*O*ut." FIFOs may be thought of as "elastic storage" devices—"logical rubber bands" between the different prts of your system, which stretch and go slack so that data rates between different subsystems do not need to match up on a short-term microsecond-by-microsecond basis, but only need to average out to be the same over a much longer period of time. This tutorial paper both describes what FIFOs are in general, and introduces the 64x4 and 64x5 Monolithic Memories FIFOs in particular.

FIFOs: Rubber-Band Memories to Hold Your System Together

Introduction

Data-rate matching problems are a very basic part of the life of a builder of digital systems. Some important electromechanical devices such as disk drives produce or absorb data at totally inflexible rates governed by media recording densities and by the speeds at which small electric motors are naturally willing to rotate. Other devices such as letter-quality printers have maximum data rates beyond which they cannot be hurried up, and which are relatively slow compared to the rates of other devices in the system.

Microprocessors and their associated main memories are generally faster and more flexible than other system components, but often operate with severly degraded efficiency if they must be diverted from their main tasks every few milliseconds to handle data-ready interrupts for individual dribs and drabs of data. While "one day at a time" may be a sound principle by which to live your life, "one bit at a time" or even "one byte at a time" is not a philosophy by which to make your microprocessor live if you want the best possible service from it.

Today there are components called "FIFOs" which let you keep your hardware design simple, and let each portion of your system see the data rate which it wants to see, and yet let you avoid hobbling the performance of your software by constantly interrupting your microprocessor, or even by intermittently halting it in order to let DMA (Direct Memory Access) circuits take over control of the main memory for a short time. FIFOs may be thought of as "elastic storage" devices — "logical rubber bands" between the different parts of your system, which stretch and go slack so that data rates between different subsystems do not need to match up on a short-term microsecond-by-microsecond basis, but only need to average out to be the same over a much longer period of time.

This tutorial paper both describes what FIFOs are in general, and introduces the 64x4 and 64x5 Monolithic Memories FIFOs in particular.

What is a FIFO?

FIFO is one of those made-up words, or *acronyms,* formed from the initials of a phrase — in this case, "First-*I*n, *F*irst-*O*ut." Originally, the phrase "First-In, First-Out" came from the field of operations research, where it describes a *queue discipline* which may be applied to the processing of the elements of any *queue* or waiting line. There is also a LIFO, or "Last-In, First-Out" queue discipline. The terms FIFO and LIFO have also been used for many years by accountants to describe formal procedures for allocating the costs of items withdrawn from an inventory, where these items have been bought over a period of time at varying prices.

You can probably think of some simple, everyday objects which in some manner behave according to the FIFO queue discipline. For instance, little two-seater cable-drawn boats are drawn through an amusement park tunnel of love one by one, and must emerge from the other end in the same order in which they entered the tunnel — "First-In, First-Out." The old-time coin dispensers used by the attendants at such amusement park features, or by city bus drivers, are "buffer storage" devices for coins which handle the coins in this same manner. (See Figure 1.)

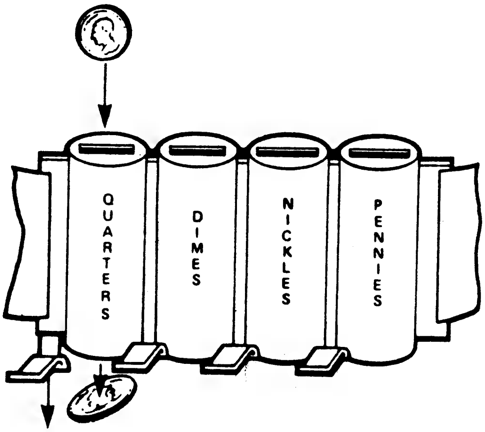


Figure 1. Primitive Mechanical FIFO Device

Notice also that the input of a coin into one of the tubes of such a coin dispenser through the slot at the top, and the output of a coin at the bottom of that tube when the lever for that tube is pushed, are completely independent events which do not have to be synchronized in any way, as long as the tube is neither totally empty nor totally full. However, if the tube fills up completely, a coin inserted into the slot will not go into the tube. Likewise, if the tube empties out completely, no coin is released from the tube at the bottom when the lever is pressed. The coin tube thus behaves as an *asynchronous* FIFO. Keep this homely example in mind.

In computer technology, both the FIFO queue discipline and the LIFO queue discipline are frequently used to control the insertion and withdrawal of information from a buffer memory, or from a dedicated buffer region of some larger memory. In input/output programming practice, a FIFO memory region is sometimes referred to as a *circular buffer,* and in programming for computer-controlled telephone systems it is called a *hopper.* A LIFO memory region is usually referred to as a *stack.*

Both FIFO and LIFO memories have frequently been implemented as special-purpose digital systems or subsystems, but as of the present time only FIFO memories are commonly implemented as individual, self-contained semiconductor devices.

Representative FIFOs

To give you the flavor of what these semiconductor devices are like, I'll describe the type 67401 64x4 FIFO and type 67402 64x5 FIFO which have been available for several years from Monolithic Memories. ("64x4" here means containing 64 words of 4 bits each.) These parts have a basic, easy-to-understand architecture and control philosophy. They also happen to be the fastest FIFOs available through normal commercial channels as of this writing, and they are in widespread use for applications ranging from microcomputers up to IBM-lookalike mainframes and large special-purpose military radar processors. A 67401 is internally organized as follows:

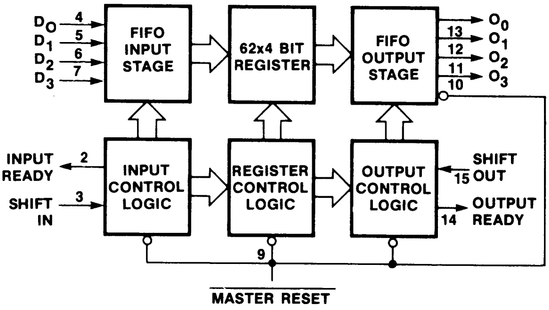
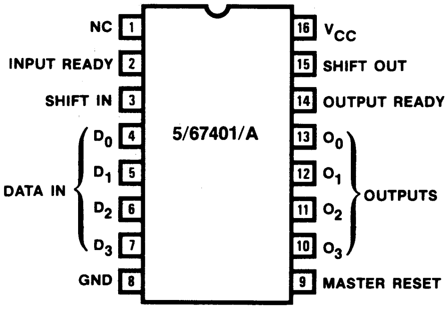


Figure 2. Architecture of the 67401 FIFO

The list of signals/pins for the 67401 is:

TYPE	HOW MANY	(CUM.)	I/O/V
Data In	4	4	I
Output	4	8	O
Control:			
Shift In	1	9	I
Shift Out	1	10	I
Master Reset	1	11	I
Status:			
Input Ready	1	12	O
Output Ready	1	13	O
Not Connected	1	14	—
Voltage:			
V_{CC} (+5V)	1	15	V
Ground	1	16	V

The corresponding list for the 67402 differs only in that there are 5 Data In lines rather than 4, and 5 Output lines rather than 4. The reason that there is an unused pin is that the 67401 was originally designed as a faster bipolar upgrade of a MOS part, the Fairchild 3341, which needs a second power-supply voltage (-12V) as well as V_{CC}. Much of the description to be given here of the 67401 also applies to the 3341, except for data rate — the 67401 can operate at 10-15 MHz depending on the exact version, compared with approximately 1 MHz for the 3341. Pinouts are:



(Note: "NC" pin is -12V for 3341.)

Figure 3. 67401/3341 Pinout

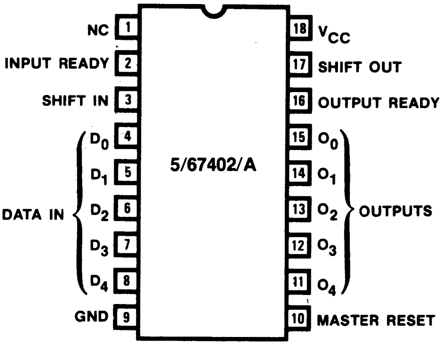
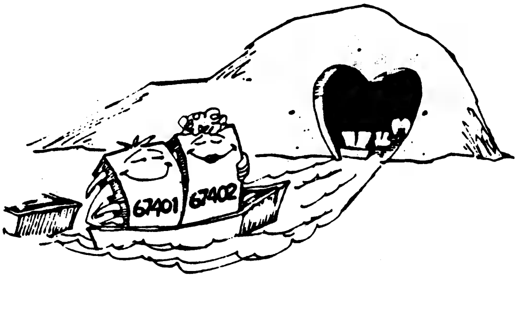


Figure 4. 67402 Pinout



"'FIRST-IN, FIRST-OUT' . . . DESCRIBES A QUEUE DISCIPLINE WHICH MAY BE APPLIED TO THE PROCESSING OF THE ELEMENTS OF ANY QUEUE . . ."

The reason for having a 5-bit model as well as a 4-bit model of basically the same part is that if two 4-bit FIFOs are placed side-by-side they make only an 8-bit FIFO, and many people have FIFO applications which entail using a parity bit with each byte and/or a frame-marker bit with the last byte of a frame or block, which means that they want 9-bit or 10-bit FIFOs. A 67402 next to a 67401 makes a 9-bit FIFO, and two 67402s make a 10-bit FIFO. But I'm getting ahead of myself.

A logic HIGH signal on the Input Ready line indicates that there is at least one vacant memory location within the FIFO into which a new data word may be inserted. Likewise, a logic High on the Output Ready line indicates that there is at least one data word currently stored within the FIFO and available for reading at the outputs. The operation of the FIFO is such that, once a data word has been inserted at the Data In lines (the *top* of the FIFO, as it were), this word automatically *sinks all the way to the bottom* (assuming that the FIFO was previously empty) and forthwith appears at the Output lines. (Remember the synonym *hopper?*) In keeping with the FIFO queue discipline, the first word which was inserted is the first one available at the outputs, and additional words may be withdrawn *only* in the order in which they were originally inserted.

There is no provision for *random access* in these FIFOs, since their internal implementation uses one particular variation of shift-register technology. Each FIFO word consists of 4 (for the 67401) or 5 (for the 67402) data bits, plus a control or "presence" bit which indicates whether or not the word contains significant information. There are thus 4 or 5 data "tracks" and one presence "track" if you look at a FIFO *from a magnetic-tape* perspective. What the Master Reset input does is to clear all of the bits in the presence track, and in addition to clear the very last data word (at the "bottom") which controls the Output lines. The other 63 data words are not cleared, but it doesn't really matter; their status is like unto that of operating-system files whose Directory entries have been deleted, in that they can no longer be read out and will get written over as soon as new information comes in.

We now return to what happens when a new data word gets inserted at the "top" of the FIFO. A mark (call it a "one") is made in the presence bit for word 00, the first word. Assume now that word 01 is vacant, so that there is a "zero" in its presence bit. The internal logic of the FIFO then operates so that the data from word 00 is automatically written into word 01, the presence bit for word 01 is automatically set to "one," and the presence bit for word 00 is automatically reset to "zero." If word 02 is likewise vacant, the process gets repeated, and so forth until the same piece of data has settled into the lowest vacant word in the FIFO — the next lower word, and all the rest, have "ones" in their presence bits, blocking further changes.

Conversely, now assume that at the moment no data word is being input, but that one has just been output. Then the bottom word in the FIFO — word 63 — has a "zero" in its presence bit, but there are a number of other words above it which have "ones" in their presence bits. The data in word 62 then moves into word 63 in the same manner described above, and the data in word 61 moves into word 62, and so forth, until there is no longer any word in the FIFO having a "one" in its presence bit which is above a word having a "zero" in its presence bit. The effect is that of empty locations bubbling up to the top of the FIFO. Or, in case you are one of those elite individuals who has been exposed to the concepts and jargon of modern semiconductor theory, you may prefer to think of the FIFO operation as one in which data ("electrons") flow from the top of the FIFO to the bottom, and vacancies ("holes") flow from the bottom of the FIFO to the top. In the general case, of course, new data words are being input at the top and old ones are being output at the bottom at random times, and there is a dynamic and continually changing situation within the FIFO as the new data words drop towards the bottom and the vacancies bubble up towards the top, and they intermix along the way.

An obvious consequences of this manner of operation in shift-register-technology FIFOs is that it takes quite a bit longer for a data word to pass all the way through the FIFO than the minimum time between successive input or output operations. There are various versions of the 67401 and 67402, rated at 7, 10, or 15 MHz over commercial (0° C to +75° C) or military (-55° C to +125° C) temperature ranges. Thus, for instance, a 15-MHz FIFO can input data words at the top and/or output data words at the bottom at a sustained rate of a word every 66-2/3 nanoseconds. However, the "fall-through time" t_{PT} for these same FIFOs is stated in the data sheet as 1.6 microseconds, which is a long enough time for 24 words to be input or 24 words to be output! There is in principle also a "bubble-through" time for a single vacancy to travel from word 63 all the way back to word 00, which should be identical to t_{PT}, and probably is although as *measured* on a semiconductor tester it may differ by as much as 50 nanoseconds, which is probably due to artifacts of measurement. By the way, the stated operating frequencies and the t_{PT} value are "worst-case" (guaranteed) numbers; the "typical" values observed in actual parts are necessarily somewhat better, since semiconductor manufacturers are obliged to take any parts back which customers can prove do not meet the worst-case numbers, and some margin of safety is always nice (see reference 1).

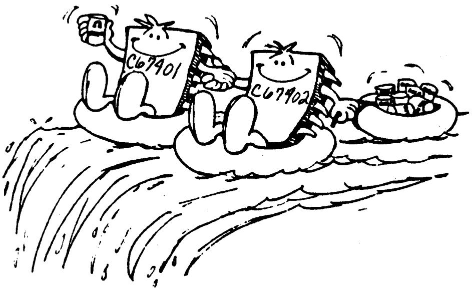
Besides Monolithic Memories, other manufacturers of bipolar (fast) FIFOs include Fairchild Semiconductor, Texas Instruments, and TRW LSI Products. MOS (slow) FIFOs are available from Advanced Micro Devices, Fairchild Semiconductor, Texas Instruments, Western Digital, Zilog, and probably other firms. FIFOs in development or available at just about all of these vendors also offer new bells and whistles which I haven't discussed, such as three-state outputs, serial (one-bit-at-a-time) as well as parallel data ports, and additional status flags. TRW's new FIFO, for instance, has a "half-full" flag which tells when half of the FIFO's words contain data. Monolithic Memories has a FIFO in development which supplies not only this flag, but also a second flag which indicates that the FIFO is either "almost full" (within 8 words of full) or "almost empty" (within 8 words of empty), reminiscent of the "yellow warning interrupt" in Digital Equipment Corporation PDP-11 computers. This "almost-full/empty flag" can be used as an interrupt to a microprocessor to indicate that *some* action must be taken, and the microprocessor can then examine the "half-full flag" to see what it actually has to do.

There are also other design approaches to the insides of a FIFO besides the one based on shift-register technology which has been described here. For instance, a FIFO may be organized as a random-access memory ("RAM") with two counters capable of addressing the RAM right within the chip, an "in-pointer" and an "out-pointer." The counting sequences, of course, "wrap around" from the highest RAM address back to zero. The out-pointer chases the in-pointer, the region just traversed by the in-pointer but not yet by the out-pointer contains significant data, and the complementary region is logically "empty." This approach involves good news and bad news: the good news is that the long fall-through time goes away, but the bad news is that now reading and writing typically interfere with each other — unless the RAM is "two-port," they cannot be done simultaneously at all. Also, since this approach is more costly in "silicon area" than the shift-register approach, it would not result in as large FIFO capacities for the same size die or the same power consumption. In practice, this approach has only been used for MOS FIFOs which have turned out to be quite slow.

Another design approach is somewhat intermediate between the pure RAM approach as just described and the shift-register approach. It uses "ring counters" on the chip instead of full-blown binary counters. What this means in practice is that there are now *two* extra "tracks" along with the data tracks within the FIFO, plus also an input data bus and an output data bus. Single "one" bits move along the in-pointer track and the out-pointer track, and the out-pointer chases the in-pointer as before. The RAM is effectively two-port, and the two parallel buses both go to each and every word. Texas Instruments has announced some small (16x4) bipolar FIFOs based on this technical approach. Like the pure RAM approach, it gets rid of the fall-through time but needs proportionally more silicon area to store a given number of bits.

Designing with FIFOs

Returning now to the Monolithic Memories 67401 and 67402, if what you *really* need is a "deeper" FIFO, say 128x4 instead of just 64x4, these parts are designed to *cascade* using a simple "handshaking" procedure, without any external logic at all! If FIFO B follows FIFO A in the cascading sequence, the Shift In control input of FIFO B is connected to the Output Ready status output of FIFO A, and likewise the Shift Out control input of FIFO A is connected to the Input Ready status output of FIFO B, and the Master Reset control inputs are all tied together. (See Figure 5.) That's all there is to it. Any number of FIFOs may be cascaded in this manner.



"...THE MONOLITHIC MEMORIES 67401 AND 67402...ARE DESIGNED TO CASCADE USING A SIMPLE "HANDSHAKING" PROCEDURE..."

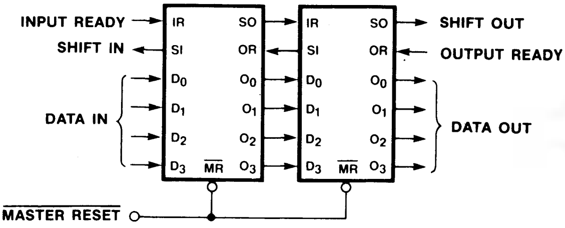


Figure 5. Cascading FIFOs to Form 128x4 FIFO

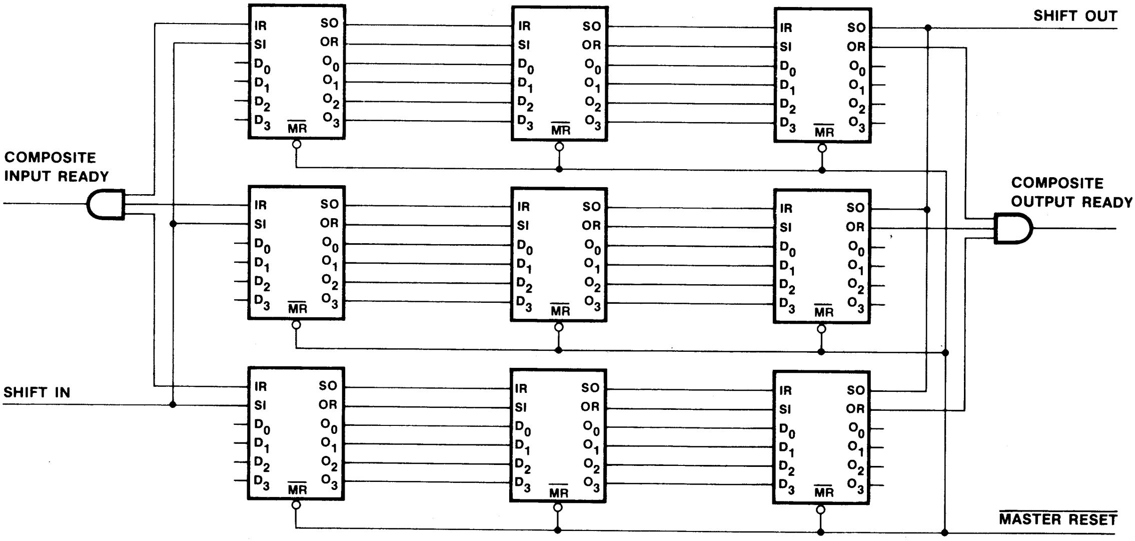
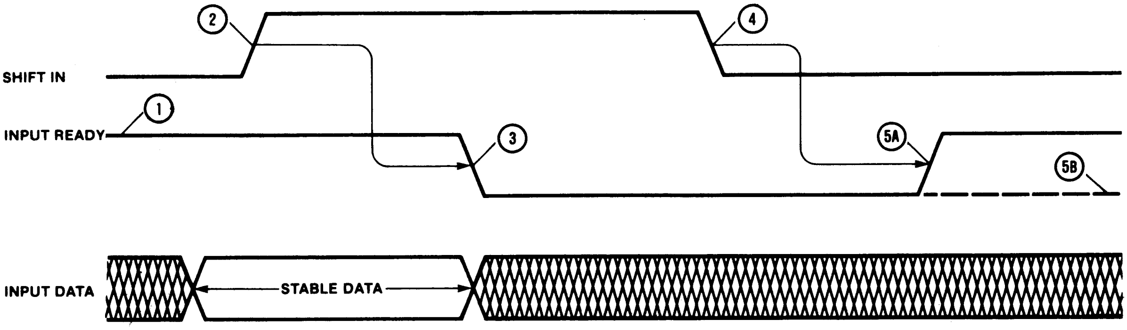


Figure 6. 192x12 FIFO

If what you *really* need is a "wider" FIFO, then you simply arrange 64x4 or 64x5 FIFOs side-by-side up to the required width. Then, you use an external AND gate such as a 74S08 or 74S11 to AND the Input Ready signals of the first rank of FIFOs if there is more than one rank, or of the only rank of FIFOs if there isn't. (See Figure 6.) Likewise, a similar AND gate is also needed to AND the Output Ready signals of the last rank of FIFOs. If you didn't provide these AND gates and just took the Input Ready signal of one FIFO as representative of when the whole array was ready, you would be taking the rather large gamble that you had correctly chosen the slowest row in this array — and if you chose wrong, 4-bit or 5-bit chunks of your input word might not get read correctly into the FIFO where they were supposed to go. Ditto on the output side. So like use the AND gates.

Although a humungus number of 67401s and 67402s are in use world-wide giving hassle-free service, it should be kept in mind that these devices are *asynchronous sequential circuits.* (One definition of "asynchronous sequential circuit" is "a fortuitous collection of race conditions," but that definition is unduly sardonic for very carefully designed parts such as these.) If your board is subject to noise, or if certain data sheet setup-time and hold-time conditions are occasionally not met, errors may occur. It is prudent system-design practice to every so often allow an array of FIFOs to empty out completely, and then issue a $\overline{\text{Master Reset}}$. (I'm assuming, of course, to start with that you're not the kind of turkey who has to be *told* to issue a $\overline{\text{Master Reset}}$ as part of your power-up sequence.) In the event that you still get what appear to be occasional errors, very small (say from 22 to 68 picofarads) capacitors from both the Shift In control input and the Shift Out input of a FIFO to ground will often eliminate these. But by all means start with a *good* circuit board — these are high-speed-Schottky-technology circuits, and like to see a lot of ground-plane metal on the board, along with other reputable interconnection practices such as 0.1-microfarad disk capacitors between V_{CC} and ground for each chip to bypass switching noise.



(1) Input Ready HIGH indicates space is available and a Shift In pulse may be applied.

(2) Input Data is loaded into the first word.

(3) Input Ready goes LOW indicating the first word is full.

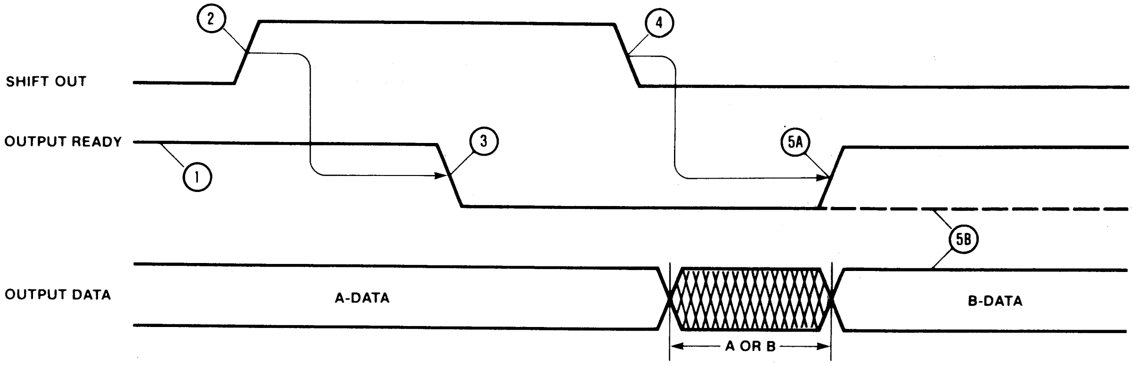
(4) The Data from the first word is released for "fall-through" to second word.

(5A) The Data from the first word is transferred to second word. The first word is now empty as indicated by Input Ready HIGH.

(5B) If the second word is already full then the data remains at the first word. Since the FIFO is now full Input Ready remains low.

Figure 7. Sequence of Events When a Data Word is Shifted into a FIFO

The sequence of events which occurs during the operation of shifting a new data word into the "top" of a FIFO is shown in Figure 7, and the corresponding sequence of events for shifting out the bottom word is shown in Figure 8. In both of these figures, it has been assumed that the external logic — whether it be the rest of your system, or just another FIFO — refrains from raising the respective Shift line to "High" until the respective Ready line has gone "High;" if the Shift line is raised any earlier, it simply gets ignored.



(1) Output Ready HIGH indicates that data is available and a Shift Out pulse may be applied.

(2) Shift Out goes HIGH causing the next step.

(3) Output Ready goes LOW.

(4) Contents of word 62 (B-DATA) is released for "fall through" to word 63.

(5A) Output Ready goes HIGH indicating that new data (B) is now available at the FIFO outputs.

(5B) If the FIFO has only one word loaded (A-DATA) then Output Ready stays LOW and the A-DATA remains unchanged at the outputs.

Figure 8. Sequence of Events When a Data Word is Shifted Out of a FIFO.

When two FIFOs are cascaded as shown in Figure 5, the sequences of events shown in Figures 7 and 8 are subject to the additional ground rule that the Output Ready line of the FIFO on the left in Figure 5 (call it "FIFO A") is identically the Shift In line of the FIFO on the right (call it "FIFO B"). And likewise, the Input Ready line of FIFO B is identically the Shift Out line of FIFO A. In the terminology we have been using, FIFO A is the "upper" FIFO and FIFO B is the "lower" FIFO. Although you do not normally need to be concerned about what happens when two FIFOs are hooked together for cascaded operation in this manner, since the "handshake" occurs quite automatically without the rest of your logic having to do anything to make it happen, it is an illuminating exercise to consider Figures 7 and 8 together in this light and see why the cascading works.

In the general case, both FIFO A and FIFO B are neither completely full nor completely empty. Thus, from the description already given of FIFO internal operation, after some period of time there will be a significant piece of data in word 63 or FIFO A and a "one" in the presence bit for that word. Since the word-63 presence bit is what controls the Output Ready signal, the latter will at some point in time go "High," and at that same point in time the data word in FIFO A word 63 is present at the output lines. Likewise, after some period of time there will be a vacancy in word 00 of FIFO B, and a "zero" in the presence bit for that word which in turn results in the Input Ready signal going "High." Remembering now that each of these Ready signals is in fact the respectively-opposite Shift signal for the other FIFO, it may be seen from Figure 7 that the conditions for inputting a word into FIFO B have now been met, and from Figure 8 that the conditions for outputting a word from FIFO A and allowing the next available piece of data from somewhere further "up" in FIFO A to enter FIFO A word 63 have also been met. The time delays shown in both Figure 7 and Figure 8 from the event at 2 to the event at 3, and from the event at 4 to the event at 5A, are asynchronous internal-logic-determined times of the order of 4 or 5 gate delays, where the gates in question are high-speed-Schottky LSI *internal* gates and have significantly less propagation delay than the SSI gates you can read about in data sheets.

After a single data word has made it across the interface from FIFO A into FIFO B, each FIFO from then on behaves in accordance with the operating rules already described, with the exact sequence of events depending on the rate at which new data words are input into FIFO A and the rate at which old data words are withdrawn from FIFO B. The net effect is that the combination of FIFO A and FIFO B with this hookup behaves almost exactly like a single integrated 128x4 FIFO. In fact, the "handshake" timing/control sequence for getting a data word from FIFO A across FIFO B is almost a replica of that which occurs within each FIFO, when the internal logic associated with word n interfaces with that associated with word n+1 for the purpose of allowing a data word to advance from occupying word n to filling a vacancy in word n+1.

Returning now to applying the timing analysis shown in Figures 7 and 8 to the case of FIFO A and FIFO B operating in cascaded mode, notice that each movement (rising or falling) of the Ready signal for one FIFO is activated by the movement in the opposite sense (falling or rising, that is) for the Ready signal from the other part. The two signals, ORA/SIB (meaning "Output Ready A" which is the same signal as "Shift In B") and IRB/SOA, cannot both remain High at the same time for more than a few nanoseconds, since if they are both High a data word will pass between the two FIFOs as already described. So, at the point when *both* the sequence of events shown in Figure 7 and the sequence of events shown in Figure 8 have been completed, and consequently ORA/SIB and IRB/SOA have both gone High again, another similar sequence of events occurs for both FIFOs and another word is passed, and so forth. This process continues apace until either ORA/SIB sticks Low, which can happen if FIFO A gets completely emptied out of data words and has "zeroes" everywhere in its presence track; or until IRB/SOA sticks Low, which can likewise happen if FIFO B gets completely filled and has "ones" everywhere in its presence track. When such a deadlock situation occurs, it lasts until a new data word has been input into FIFO A and has had time to "fall all the way through" and settle into FIFO A word 63, or until the data word in word 63 of FIFO B has been read out and the resulting vacancy has had time to "bubble all the way back up" into FIFO B word 00, as the case may be.

Various Uses for FIFOs

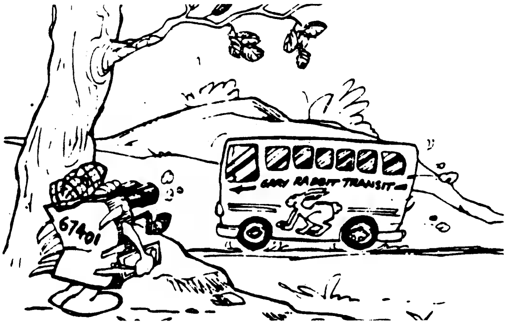
The classical FIFO application, as already mentioned at the beginning of this paper, is that of matching the instantaneous data rates of two digital systems in a simple, economical way. One of the two systems may, for reasons of design economics or even of utter necessity, want to emit or absorb data words in ultra-high-speed bursts, whereas the other one may prefer to operate at a slow-but-steady data rate or even at an erratic rate which varies between ultra-slow and slow or even between slow and fast. No matter — it's all the same to an asynchronous FIFO such as the 67401 or 67402, as long as the input rate and the output rate do match up over a long period of time so that it neither fills up nor empties out.

There are, however, some additional uses for FIFOs which arise from other, rather different circumstances. For instance, your digital system may simply need some extra buffer storage scattered around locally at different points on your block diagram, and you and your system may really just not care whether this storage is accessed on a random or on a queue basis. Under these circumstances, it is ordinarily less hassle to use a FIFO than to use a small RAM and come up with some extra logic to generate addresses and timing signals for it. Often the FIFO modus operandi is in fact the natural one for the application; as for instance when your system must accumulate a block of 64 characters and then run them by all at once in order to examine them for the presence of some control character, using some scanning logic — or perhaps even a microprocessor — which is otherwise occupied most of the time.

A less obvious but interesting application of FIFOs is as automatic "bus-watchers" for jump-history recording for hardware or even software diagnostic purposes. A FIFO whose inputs are connected to a minicomputer's program counter or microprogram counter, or to a microcomputer's main address bus, may be operated so as to record every new jump address generated by the program. This way, if at some point the hardware freaks out or the operating system crashes, a record exists of the last 64 jumps which were taken before the system was halted, assuming of course that you have provided some way for the system to sense that all is not well and halt itself. Such a record of jumps can be very valuable in tracing out what happened just before everything went haywire. FIFOs may be used in this way either as part of built-in self-monitoring features in digital systems, or as part of various kinds of external test equipment.

FIFOs may also be used as controllable delay elements for digital information which cannot be used immediately upon receipt — perhaps it must be matched against other information which is not yet available, or perhaps it must be synchronized with other streams of information which are out of phase by a varying amount. An example of the latter situation is deskewing several bit-streams off a parallel-format magnetic tape, which commonly has to be done when high recording densities are used. One FIFO *per bit-stream* is required — but the net resulting logic may still be the most reliable and economical way to get the job done, when compared with other possible digital designs. Another example is that of using FIFOs as data memories in digital correlators; the lag in an autocorrelation operation can be set simply by controlling how many words are in the FIFO at one time, and so forth. There are even some applications in which it is advantageous to operate a FIFO with all of its input and output cycles synchronized, so that absolutely all it does is to delay the data by some certain number of clock intervals.

References (2), (3), and (4) are formal applications notes available from Monolithic Memories, which discuss FIFOs from different viewpoints than this paper has taken. Each of them presents a more detailed explanation of one or more applications than there has been room for here. Reference (2) is mainly an overall applications survey, reference (3) emphasizes digital communications, and reference (4) emphasizes digital spectrum analyzers and also includes an overview of digital signal processing in general.



"A LESS OBVIOUS BUT INTERESTING APPLICATION OF FIFOS IS AS AUTOMATIC 'BUS-WATCHERS' . . ."

References

(1) *Bipolar LSI 1982 Databook*, Monolithic Memories, Inc., 1165 East Arques Avenue, Sunnyvale, CA 94086. Section 9 of this databook covers FIFOs.

(2) "First In First Out Memories . . . Operations and Applications," applications note published March 1978 by Monolithic Memories, Inc. [See (1) for the address.] A good survey with some thought-provoking ideas. I have to mention one error, however; one circuit diagram shows parallelled FIFO operation in the manner of that of Figure 6 in this paper, but without the use of the AND gates for the composite Ready signals. As I already warned you, that's *dangerous*.

(3) "Understanding FIFO's," applications note published by Monolithic Memories, Inc. [See (1) for the address.] The author, Alan Weissberger, has also now gotten a modified version of this note published as a magazine article, "FIFOs Eliminate the Delay when Data Rates Differ," in *Electronic Design*, November 27, 1981. Despite the general title, the emphasis is on digital communications applications.

(4) "PROMs, PALs, FIFOs, and Multipliers Team Up to Implement Single-Board High-Performance Audio Spectrum Analyzer," applications note published by Monolithic Memories, Inc. [See (1) for the address.] The author, Richard Wm. Blasco, also got this note published in *Electronic Design* in two installments, in the issues of August 20 and September 3, 1981 under the titles "PAL Shrinks Audio Spectrum Analyzer" and "PAL Improves Spectrum Analyzer Performance" respectively.

Pick the Right 8-Bit —Or 16-Bit— Interface Part for the Job*

Chuck Hastings and Bernard Brafman

A few years ago, 20-pin 8-bit buffers, registers, latches, and transceivers came into existence as a rather haphazard upwards evolution from the MSI devices available in the mid-1970s. As time went on, usage of these parts increased until they became one of the fundamental computer-system building-block "primitives"—the "glue" which holds the entire system together. More recently, there has emerged an orderly, matrix-like approach to combining useful attributes of interface circuits, such as Schmitt-trigger inputs, inverting outputs, high-drive outputs, and series-resistor outputs, into specific parts.

Today the demands are to reduce component costs and system board area. Reducing parts count achieves both of these objectives at one stroke; it is now possible to effectively incorporate the equivalent of two 20-pin 8-bit interface parts into one 24-pin "16-bit interface" part. The approach is to look for common configurations of pairs of 8-bit parts, and implement the pair as a single chip.

* This paper is a slightly modified version of the paper by the same name which appeared in the *Northcon/82 Professional Program Session Record,* Session 15 reprint, paper 15/3; 18-20 May 1982.

Pick the Right 8-Bit—or 16-Bit—Interface Part for the Job

Chuck Hastings and Bernard Brafman

Abstract

A few years ago, 20-pin 8-bit buffers, registers, latches, and transceivers came into existence as a rather haphazard upwards evolution from the MSI devices available in the mid-1970s. As time went on, usage of these parts increased until they became one of the fundamental computer-system building-block "primitives"—the "glue" which holds the entire system together. System designers demanded, and semiconductor manufacturers provided, many refinements such as inverting outputs to reduce parts count in assertive-low-bus systems, high-drive outputs to rescue designs with overloaded buses, Schmitt-trigger inputs to likewise rescue designs troubled with severe bus noise, high-voltage outputs specifically suited for driving MOS inputs, series-resistor outputs for driving highly capacitative loads such as dynamic-MOS address buses, and so forth.

Today the demands are to reduce component costs and system board area. Reducing parts count achieves both of these objectives at one stroke. With the development of the 300-mil 24-pin SKINNYDIP™ package, it is now possible to effectively incorporate the equivalent of two 20-pin 8-bit interface parts into one 24-pin "16-bit interface" part. The approach is to look for common configurations of pairs of 8-bit parts, and implement the pair as a single chip. Common configurations include back-to-back "registered transceivers," with the same options already available in the 20-pin 8-bit parts, and pipeline registers.

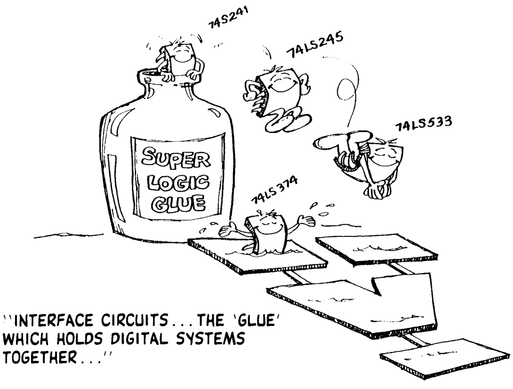
Interface Basics

Where Do Interface Circuits Fit In?

Interface circuits appear as unglamorous bread-and-butter commodity items, as compared to many of the other more complex integrated circuits of today: their sales volume is very high, their average selling price is comparatively low, and essentially interchangeable parts are offered by several suppliers. They have the humble role of being the "glue" which holds digital systems together; they are *means* rather than ends in themselves.

When preliminary system block diagrams turn into detailed schematics, the *blocks* turn into complex circuits—microprocessors, multipliers/dividers, automatic dynamic-MOSRAM refresh controllers, high-speed FIFOs, programmable-logic circuits, arithmetic-logic units, and so forth. But then, however, the *lines* between those blocks turn into interface circuits, which must be there in the final design but never explicitly get noticed during the conceptual-design stage!

The term "interface" is actually a bit of a misnomer, since it implies that these parts always occur at a boundary between two somewhat different types of logic. That may have been true once, and it is still true that many of the circuits commonly called "interface" have inputs and/or outputs which are different electrically from those of, say, triple three-input NAND gates produced using the identical solid-state-circuit technologies. But a general working definition of "interface circuits" also has to cover some other parts which get used in similar system roles, but have normal inputs and normal totem-pole or three-state outputs. One such definition, current today at Monolithic Memories, is



"INTERFACE CIRCUITS...THE 'GLUE' WHICH HOLDS DIGITAL SYSTEMS TOGETHER..."

"... ultra-high performance integrated circuits which do not lend themselves to higher levels of integration, due either to their parallel data structure or to the electrical properties of their inputs and/or outputs."

Interface circuits get used wherever data must be held, transmitted on demand, power-amplified, level-shifted, read from a noisy bus, inverted, or otherwise operated upon in some simple electrical way. If more complex transformations of the data are called for, of a predominantly mathematical rather than electrical nature, the designer will typically try to perform the required operations with readymade LSI or MSI circuits. Even here, of course, interface circuits often have the inconspicuous but crucial role of performing format conversion so that several LSI circuits can communicate with each other. Still, they are viewed as "overhead," which system designers try to minimize and semiconductor producers often rank well below their top level of corporate priorities.

But interface circuits are here to stay, at least for several more years. And the realization is growing among both users and producers of semiconductors that, since interface parts are not about to vanish soon, they need to be treated as something more than afterthoughts to the design process. Users who select interface circuits shrewdly are achieving real gains in system performance and reliability, and significant reductions in system size, weight, and power consumption. Producers who do a conscientious and professional job of developing and marketing these humble parts are finding increased demand for their wares, even during recessions.

Two major trends currently evident in the world of interface circuits are:

- **The emergence of an orderly, matrix-like approach to interface products, so that taken all together they form an *array* rather than simply a splendid jumble of assorted types.**
- **A strong emphasis on increasing the number of data bits which can be handled or accomodated by a single interface-circuit package.**

This paper will discuss each of these trends in some detail, and will then go on to present some realistic interface applications based on several actual designs.

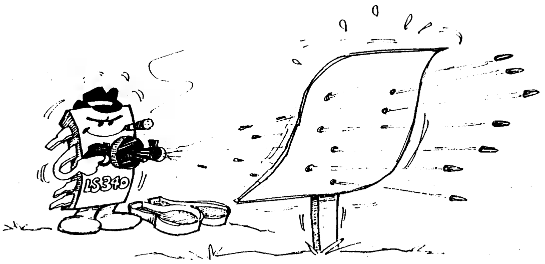
What Kinds of Interface Circuits Are There?

Commonly, the label "interface circuit" is applied to any of a diverse collection of miscellaneous devices which don't seem to fit into any other classification. As the term is used here, however, it means either one of three basic 8-bit types—buffers, latches, and registers—which are *simple* interface circuits, or else one of several 16-bit *compound* interface circuit types such as transceivers and pipelines.

Buffers merely "pass" or transmit information at increased power levels.Most contemporary buffer circuits, including 20-pin 8-bit buffers, also have an electronically-selectable electrical-isolation capability. Such a *three-state* buffer has a type of output which can be switched into a "hi-Z" (high-impedance) state in which it does not drive, nor appreciably load, the circuit node to which it is attached.

True or *noninverting* buffers pass the input information along with the same polarity (i.e., conventions in the representation of ones and zeroes by high and low voltages) that it had when it was received. *Inverting* buffers reverse the polarity of the input information from what was received, complementing all ones to zeroes and all zeroes to ones.

Most buffers feature standard PNP inputs. However, the 'LS340/341/344/310 buffers feature Schmitt-trigger inputs, with a guaranteed 400-millivolt deadband (typically 800 millivolts) centered about the switching threshold voltage. (This notation is shorthand for "SN54/74LS340, SN54/74LS341, SN54/74LS344, SN54/74LS310," and will be used frequently hereafter.) These Schmitt-trigger buffers won't respond to input noise pulses which would make buffers with normal inputs start to switch, as long as the noise pulses do not completely cross the deadband; thus noise immunity is improved.



"...THE 'LS340/341/344/310 BUFFERS FEATURE SCHMITT-TRIGGER INPUTS, WITH A GUARANTEED... DEADBAND..."

Latches and *registers* have the same basic capability as buffers, but also have the additional capability that they retain stored information as long as power is supplied to them. Each of these circuit types requires an additional control signal in order to perform its system function.

More specifically, *latches* use an *enable* signal. When this signal is on, they store information, and their outputs do not change even if the information presented to their inputs changes. When their enable signal is off, latches act just like buffers. Turning on the enable signal in effect "freezes" in place whatever information was passing through the latch, so that the latch stores it.

Registers use a *clock* signal instead of an enable signal. When the clock signal goes through a transition from off to on, this "rising edge" causes the information present at the inputs to be stored in the register, and then to remain present at the register outputs until another rising edge occurs. When the clock is in a steady-state condition (a "level"), either on or off, or even when the clock goes through a transition from on to off (a "falling edge"), the outputs of the register do not change. Thus, unlike latches, registers lack a mode in which they act exactly like buffers and pass information directly from their inputs to their outputs. This lack is a consequence of the control signal being "edge-sensitive" rather than "level-sensitive."

Transceivers are bidirectional interface circuits capable of interconnecting two buses so that information can pass in either direction. Most of the transceiver parts in production today are *buffer transceivers*—they are like two cross-coupled buffer circuits within a single 20-pin package. A 16-bit buffer transceiver has eight A-bus data pins and eight B-bus data pins. Either the A-to-B buffers may be enabled, or the B-to-A buffers, or neither; if both sets of buffers were to be enabled, obviously there would be a race condition on each of the data lines, and so the control structure of some buffer transceivers specifically disallows that mode of operation. (Some other types do allow it.) Buffers which are not enabled are, of course, in the hi-Z state. Thus each buffer transceiver interface circuit consists of eight logical elements, and each of these logical elements consists of two simple-buffer elements cross-coupled back-to-back so that the input line for one is the output line for the other and conversely.

Latch transceivers and *register transceivers* have not yet become major factors in the marketplace, but several semiconductor houses now have such devices in development. Two factors have delayed their introduction relative to that of buffer transceivers: they require too many control signals to fit into a standard 20-pin interface-circuit package, and they dissipate more power than buffer transceivers. Both of these problems have by now essentially been solved.

Pipelines are unidirectional interface circuits having more than one full-width internal latch/register or "stage," but typically having just one set of parallel data inputs and one set of parallel data outputs. Two-stage latch pipelines, and both two-stage and four-stage register pipelines, are coming soon also. The four-stage devices can store twice as much information per package, but the two-stage devices can be reconfigured more flexibly and have a greater degree of separate control for each stage.

Understanding and Using Interface

How Designers Choose Interface Circuits

In the real world, a digital-logic designer doesn't set out deliberately to use some particular interface circuit whose properties he has carefully learned, in the same way that he might for instance set out to use a bit-slice registered ALU or a multiplier/divider. Rather, as we have said, it is much more likely that it all starts with some innocent-looking little line between two blocks on his preliminary system block diagram which, it turns out, can't really be just a simple little line after all.

Maybe the data which travels on that little line goes away at the source unless the little line is actually also capable of seizing it at the proper time and remembering it. Or maybe the end of the little line is an assertive-low system bus, with enough loads hanging off it to call for almost 30 milliamps of drive capability in whatever contemplates driving the bus, which doesn't quite jibe with the 2-milliamp drive capabilities and assertive-high outputs of the MOS LSI device from which the data is coming.

Octal Interface Selection Guide

FUNCTION	POWER	POLARITY	FEATURE	PART NUMBER	
				COMMERCIAL	MILITARY
Buffer	LS	Non-invert	—	SN74LS244	SN54LS244
			—	SN74LS241	SN54LS241
			Schmitt Trigger	SN74LS344	SN54LS344
			Schmitt Trigger	SN74LS341	SN54LS341
		Invert	—	SN74LS210	SN54LS210
			—	SN74LS240	SN54LS240
			Schmitt Trigger	SN74LS310	SN54LS310
			Schmitt Trigger	SN74LS340	SN54LS340
	S	Non-invert	—	SN74S244	SN54S244
			—	SN74S241	SN54S241
		Invert	—	SN74S210	SN54S210
			—	SN74S240	SN54S240
Transceiver	LS	Non-invert	—	SN74LS245	SN54LS245
			—	SN74LS645	SN54LS645
			48mA I_{OL}	SN74LS645-1	—
Latch	LS	Non-invert	—	SN74LS373	SN54LS373
		Invert	—	SN74LS533	SN54LS533
	S	Non-invert	—	SN74S373	SN54S373
			32mA I_{OL}	SN74S531	—
		Invert	—	SN74S533	SN54S533
			32mA I_{OL}	SN74S535	—
Register	LS	Non-invert	Master Reset	SN74LS273	SN54LS273
			—	SN74LS374	SN54LS374
			Clock Enable	SN74LS377	SN54LS377
		Invert	—	SN74LS534	SN54LS534
	S	Non-invert	—	SN74S374	SN54S374
			32mA I_{OL}	SN74S532	—
		Invert	—	SN74S534	SN54S534
			32mA I_{OL}	SN74S536	—

Table 1. Example of Interface-Part-Selection Matrix

At this point the designer needs an interface circuit, and—wittingly or unwittingly—he must go through a several-stage decision process to determine what interface circuit he needs to actually implement that little line, before his block diagram can turn into a system. He must also fervently hope that, by the time he gets to the final twig on his decision tree, the interface part he needs will turn out to actually exist. Figure 1 is an example.

A top-down design approach, as illustrated in Figure 1, isn't always wise with integrated circuits, simply because the chances are fairly good that the desperately needed circuit actually *won't* exist[r1]. And there was a time, not all that long ago, when only a quasi-random subset of all of the obviously possible variations of the basic interface parts had reached full production status, so that they could be bought and plugged in. The hapless designer just had to memorize what that subset was, and do his design bottom-up from there.

Today, chaos is giving way to order, and enough of the possible interface parts which a designer might want do by now exist (or will exist shortly) that the kind of top-down thought process portrayed in Figure 1 really *will* work out all right when designing with interface. For instance, the line of interface parts now in production at Monolithic Memories is sufficiently orderly to be organizable into the matrix of Table 1. Although Table 1 is still somewhat irregular, it is at least recognizable as first-cousin to a logic-design Karnaugh map, and you can actually get your hands on any of the interface parts in the matrix.[r2]

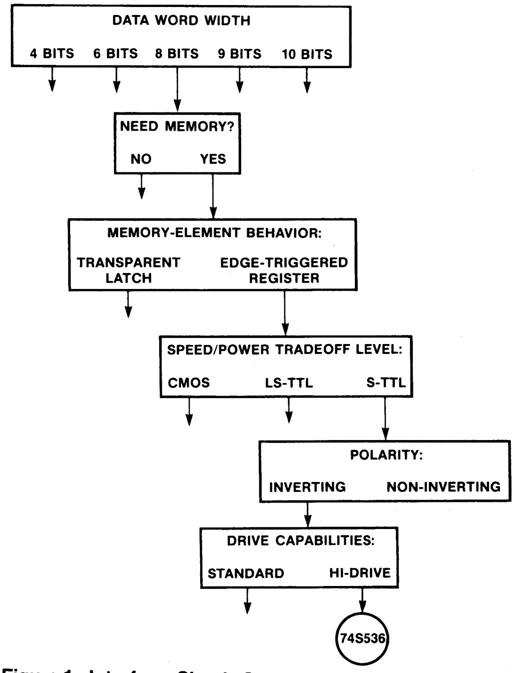


Figure 1. Interface-Circuit-Selection Decision Tree

The dimensions of variation for interface parts in any such Karnaugh map are, of course, two-valued "Boolean" variables. It is realistic from both logical and historical viewpoints to consider that all of the interface parts of Table 1 have been derived from a very few basic types, by implementing those combinations which make sense of several two-valued properties of interface parts. These are:

- Commercial versus military temperature-range operation.
- High-speed Schottky (S-TTL) or low-power Schottky (LS-TTL) speed/power range.
- Noninverting or inverting outputs.
- No memory capabilities in the logical elements, so that they operate as buffers; or memory capabilities therein, further subdivided according to whether the logical elements operate as latches or registers.
- Compound 16-bit interface circuits or simple 8-bit interface circuits.
- Hi-drive or standard levels of current-sinking capability (I_{OL}) at the outputs.
- Schmitt-trigger or standard inputs.
- For non-three-state parts, master-reset or clock-enable control inputs.
- (In versions of this chart yet to be issued) Series-resistor or standard outputs.

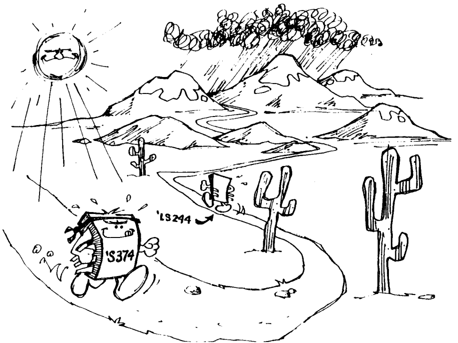
Obviously, not all imaginable combinations of the above properties actually exist as parts, or would even be useful if they did; and semiconductor houses cannot afford for long to offer 2^n interface-circuit part types for rapidly increasing n. Moreover, certain of the properties which today have just two possible major choices (e.g., S-TTL and LS-TTL) may soon have more than two.

Nevertheless, by now the matrix approach has been fully-enough implemented to offer a very helpful perspective to the working designer.

Part numbers today allow some of the properties of interface circuits to be directly inferred, at least if the part number follows the conventions of the industry-standard "54/74" numbering series. 54/74 part numbers have a well-defined format VVE4TxxxP, with the following interpretation:

- VV — a prefix which varies somewhat from vendor to vendor, although several vendors now use the prefix "SN."
- E4 — a temperature-range environmental specification. "54" implies the military temperature range (−55°C to +125°C), and "74" the commercial temperature range (0°C to +70°C for several vendors, and 0°C to +75°C for Monolithic Memories). In any case, interface circuits must run properly over a very wide temperature range.
- T — a solid-state-circuit technology. Upwards of a dozen of these have been promoted, with widely varying success, during the last decade. The earliest one, plain old gold-doped TTL, omitted using any special letter in part numbers. Today, the two dominant technologies are "S" (high-speed Schottky) and "LS" (low-power Schottky). Others likely to become quite important include "F" (for "FAST," a lower-power form of high-speed Schottky), "ALS" (advanced low-power Schottky), and "SC" (isoplanar CMOS processed to be fully TTL-voltage-level compatible).
- xxx — a two-digit, three-digit, and today sometimes even four-digit number which uniquely specifies the *pin-out* of the part and its "functional behavior" (see the explanation which follows), independent of speed/power range.
- P — a package type: plastic, cerdip, flatpack, leadless chip carrier, sidebrazed ceramic, or whatever.

The functional behavior of a circuit can be defined somewhat circularly as **"what a designer needs to know about the circuit in order to construct designs which operate properly using parts from any supplier interchangeably."** This definition is akin to one classic definition of computer architecture as **". . . the structure of the computer a programmer needs to know in order to be able to write any program that will correctly run on the computer."**[r3]



". . . INTERFACE CIRCUITS MUST
RUN PROPERLY OVER A VERY WIDE TEMPERATURE RANGE . . ."

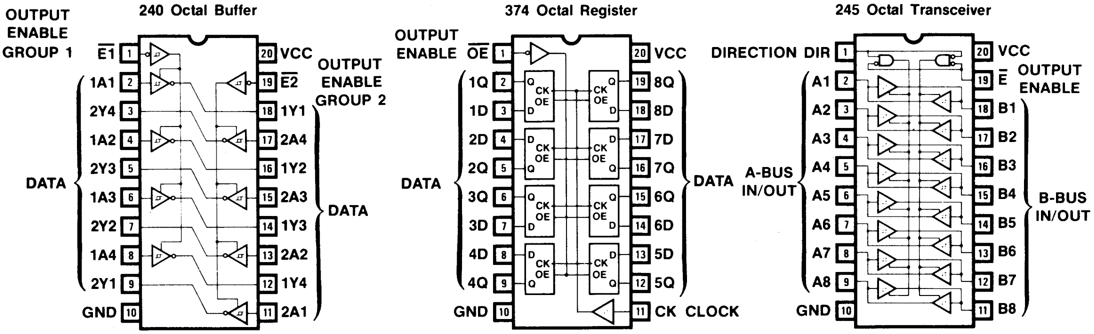


Figure 2. Pinouts for the Three Basic 20-Pin Interface Parts

Two parts produced using different solid-state-circuit technologies may exhibit essentially the same functional behavior. If that is the case, and if either part will also satisfy system timing constraints (which is an issue quite separate from that of "functional behavior") and input/output voltage compatibility constraints, the designer does not need to care what kind of internal gates are used within the part—Schottky TTL, ECL, CMOS, NMOS, or water wheels. On the other hand, two parts produced using the same technology may have subtle, or even drastic, differences in their functional behavior; for example, one may have inverting outputs, or hi-drive outputs, or Schmitt-trigger inputs whereas the other does not.

The Matrix of Interface Part Types

The interface parts of Table 1 all have one of just three different pinouts, shown in Figure 2, in their usual 20-pin plastic or cerdip SKINNYDIP™ form.

All of the buffers have the same pinout as the 'S240. They differ in speed/power range, in the polarity of the outputs, in the noise-rejection capabilities of the inputs (Schmitt-trigger or standard), and in enable structure (complementary or assertive-low) as shown in Figure 3, which really *is* unequivocally a Karnaugh map.

	COMPLEMENTARY ENABLES		ASSERTIVE-LOW ENABLES		
	INVERTING OUTPUTS	NON-INVERTING OUTPUTS		INVERTING OUTPUTS	
S-TTL	'S210	'S241	'S244	'S240	USUAL INPUTS
	'S310**	'S341*	'S344*	'S340*	SCHMITT-TRIGGER INPUTS
LS-TTL	'LS310	'LS341	'LS344	'LS340	
	'LS210	'LS241	'LS244	'LS240	USUAL INPUTS

NOTES: *— available from Texas Instruments, and planned by Monolithic Memories.
**— planned by Monolithic Memories.

Figure 3. 8-Bit Three-State Buffers

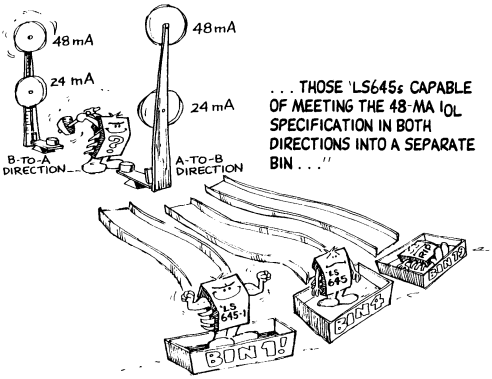
All of the latches and registers have the same pinout as the 'S374. They differ in whether the memory control line is level-sensitive (latch) or edge-sensitive (register), in speed/power range, in the polarity of the outputs, and in the I_{OL} (current-sinking drive) capability of the outputs as shown in the Karnaugh map of Figure 4.

	LATCHES		REGISTERS		
	INVERTING OUTPUTS	NON-INVERTING OUTPUTS		INVERTING OUTPUTS	
S-TTL	'S533	'S373	'S374	'S534	USUAL (20-mA) OUTPUTS
	'S535	'S531	'S532	'S536	HI-DRIVE (32-mA) OUTPUTS
LS-TTL	— —	— —	— —	— —	
	'LS533	'LS373	'LS374	'LS534	USUAL (24-mA) OUTPUTS

Figure 4. 8-Bit Three-State Latches and Registers

The three transceivers of Table 1 are more specifically *buffer transceivers*—compound 16-bit interface circuits like two 8-bit buffer circuits cross-coupled "back-to-back" within a single device. They differ in input-current and output-leakage-current specifications, which here are indistinguishable for test purposes since every data pin is both an input and an output; the 'LS245 specification is tighter. (The 'LS245 is *also* specified as faster, but that is *not* a difference in "functional behavior.") There is also a difference in I_{OL} capability; the 'LS645-1 is specified as higher. Actually, all three devices undergo identical fabrication, and are separated only at final testing; for instance, those 'LS645s capable of meeting the 48-mA I_{OL} specification in both directions drop into a separate bin.

Upcoming developments in interface parts will tend in many cases to follow the matrix approach, at least partially. Even where the new parts do not fit perfectly into the matrix of existing parts, some attention is likely to be paid to issues of balance and symmetry over the entire interface-circuit product line.



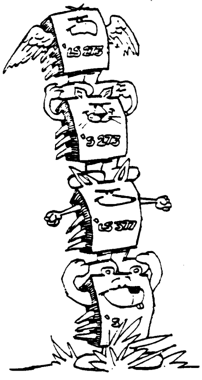
". . . THOSE 'LS645s CAPABLE OF MEETING THE 48-MA I_{OL} SPECIFICATION IN BOTH DIRECTIONS INTO A SEPARATE BIN . . ."

In some cases, new interface parts directly "fill in the holes" in the matrix. For instance, the next planned additions to Monolithic Memories' line of interface parts are:

Function	Speed/ Power	Polarity	Feature	Part Number
Register	S	Noninv.	Master Reset	SN54/74S273
Register	S	Noninv.	Clock Enable	SN54/74S377
Buffer	S	Noninv.	Series Output Resistor	SN54/74S734*
Buffer	S	Noninv.	Series Output Resistor	SN54/74S731
Buffer	S	Inv.	Series Output Resistor	SN54/74S730#
Buffer	S	Inv.	Series Output Resistor	SN54/74S700

NOTES: *—The 'S734 is a direct replacement for AMD's Am2966.
#— The 'S730 is a direct replacement for AMD's Am2965.

Table 2. Pending Additions to the Monolithic Memories Interface-Part-Type Matrix



"... THE 'S273 AND 'S377, LIKE THEIR LS-TTL COUNTERPARTS, ARE DESIGNED WITH STANDARD TTL 'TOTEM-POLE' OUTPUTS..."

The 'S273 and 'S377 bring to higher-performance TTL systems the same functional behavior which has long been available for medium-performance TTL systems, with the popular 'LS273 and 'LS377 parts. The 'S273 and 'S377, like their LS-TTL counterparts, are designed with standard TTL "totem-pole" outputs. Somehow, in the somewhat more chaotic early days of 8-bit interface, the need for high-speed Schottky versions of these parts got overlooked by most interface producers.

Since the 'S273 and 'S377 are totem-pole-output parts, the control pin which gets used on the 'S374 (whose pinout they otherwise follow) for "Output Enable" for the three-state outputs is available for something else. The 'S273 uses it as a "Master Reset" ($\overline{MR}$) input, capable of forcing all of the eight D-type flipflops on the chip into the off (low) state simultaneously, regardless of their previous state—*or* of the state of the clock line and/or the data-input lines. The 'S377, on the other hand, uses that same pin as a "Clock Enable" ($\overline{CK\ EN}$) input, which in effect either allows the clock signal to reach the eight D-type flipflops on the chip, or else cuts it off from reaching the flipflops so that they are not clocked and just sit there holding whatever information they contained previously.

The major applications for these parts are in situations where 'S374s would be difficult to control appropriately. Because of the 'S273's $\overline{MR}$ input, its forte is *control* applications—instruction registers, microinstruction registers, timing-pulse registers, and sequential circuits in general, and sometimes as eight *individual* separate D-type control flipflops in one package. In all of these applications, there *has* to be a way to force the system into some proper initial state, so that it "starts off on the right foot" and does not get into some unplanned-for, untestable, unpredictable machine-psycho condition on power-up. The 'S377, on the other hand, because of its $\overline{CK\ EN}$ input, is the optimum choice for the highest-performance TTL pipeline paths for data, instructions, microinstructions, and address parameters in "overlapped-architecture" machines such as array processors and high-performance minicomputers.

The 'S700, 'S730, 'S731, and 'S734 feature a new type of output stage incorporating a series resistor, designed to efficiently drive highly-capacitative loads such as arrays of dynamic-MOSRAM inputs. Rise and fall times are more symmetric than with 'S240-type buffers, and the latter need an *external* series limiting resistor for their own protection when driving highly capacitative loads.

Consequently, although 'S240-type buffers may exhibit greater speed when tested under light loading conditions, 'S730-type buffers are likely to perform better under realistic system conditions when driving large distributed capacitative loads is a major factor in the application.

Of these four new buffers, two—the 'S730 and 'S734—are second-source versions of the Am2965 and Am2966 respectively, originally introduced by AMD. The other two—the 'S700 and 'S731—are complementary-enable versions of the 'S730 and 'S734 respectively, just as the 'S210 and 'S241 are complementary-enable versions of the 'S240 and 'S244 respectively. Complementary-enable buffers excel in driving buses with two multiplexed sources for the information, such as instruction addresses and data addresses in a bit-slice bipolar microcomputer system.

The four new 'S730-type parts may be grouped with Monolithic Memories' existing line of buffers in a 2 × 2 matrix chart or Karnaugh map, with the dimensions of this map chosen to be the polarity of the second-buffer-group enable input E_2 (here across the top) and the polarity of the data-buffer logical elements themselves (here down the side), thus:

		Polarity of E_2*	
		$\overline{E}_2$	E_2
Polarity of Data Buffers	Inverting	'LS240 'LS340 'S240 'S730	'LS210 'LS310 'S210 'S700
	Noninverting	'LS244 'LS344 'S244 'S734	'LS241 'LS341 'S241 'S731

*NOTES:**— Since $\overline{E}_1$ is assertive-low for *all* of these parts, the parts with an assertive-low $\overline{E}_2$ are "assertive-low enable" parts, whereas the parts with an assertive-high E_2 are "complementary-enable" parts.

Table 3. 8-Bit Buffers Grouped by Polarity and Enable Structure

By this time, many presently-unused SN54/74xxx part numbers have already been reserved for other potential new parts, even though not all of these parts are yet in production. Nevertheless, it was at least possible to part-number these four series-output-resistor buffers in such a way that the relationship among the four types remains the same as for 'S240-type buffers. To state this another way, one can add 490 to the last three digits of the usual buffer part number to get the part number for the corresponding series-output-resistor part, e.g., 'S241 + 490 = 'S731, etc.

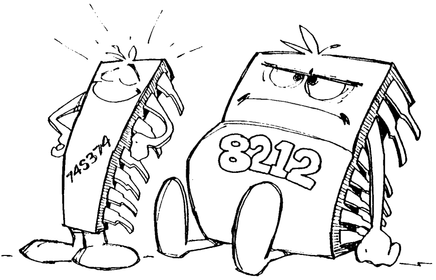
Directions In The Evolution of Interface Parts

More Bits per Package

Historically, the first interface parts were 16-pin TTL devices offered during the early 1970s, usually with four or six "logical elements" per package. One "logical element" handles one data bit; in simple interface parts, a logical element may be a buffer, a latch, or a register (with "register" here implying an edge-triggered flipflop).

As the digital-electronics industry shifted from MSI to LSI integrated circuits, and from the quaint and irregular old-time computer word lengths to word lengths which are multiples of eight bits (most often 8, 16, or 32), 8-bit interface devices became the only way to go for simple electrical data transformations—chip counts got intolerably high with 4-bit devices, and 6-bit devices were awkward misfits in most of the newer designs.[r4] And, to have eight input data lines, eight output data lines, power and ground, and two control signals, an integrated-circuit package has to have 20 pins.

To conserve board space, the width of this 20-pin package was chosen to be 300 mils (.300") like that of the overwhelming majority of the then-existing bipolar MSI and SSI devices. Hence, during the 1970s, the present 20-pin 300-mil SKINNYDIP™ package became the standard for interface circuits. One 20-pin SKINNYDIP™ takes up only about half as much board space as one of the older 600-mil 24-pin packages, which were then being used for a few early 8-bit interface parts such as the Intel 8212.



"...ONE 20-PIN SKINNYDIP™ TAKES UP ONLY ABOUT HALF AS MUCH BOARD SPACE AS ONE OF THE OLDER 600-MIL 24-PIN PACKAGES..."

24-pin interface parts appear to be the next major development to come. In the early 1980s, mechanical packaging problems which previously had inhibited the introduction of a 24-pin 300-mil SKINNYDIP™ were solved, and this package is now also coming into widespread use for PROMs, PAL® programmable-logic circuits, and so forth. So what might one do with four additional pins in an interface part?

One answer is to spend all four of them for additional *control* signals in order to achieve more flexible parts, such as the new Monolithic Memories SN54/74LS380 "multifunction" 8-bit register.[r2] This part is actually implemented with "hard-array logic" technology, and has an internal structure like one form of PAL®.

Another answer is to spend all four of them for additional *data* signals, equally for inputs and outputs. The result is 10-bit interface parts with functionality similar to that of existing 20-pin 8-bit parts.

A middle-of-the-road answer is to divide them equally between control signals and data signals. This approach leads to 9-bit interface parts with improved functionality.

16-bit interface circuits—dual 8-bit circuits in a single 24-pin SKINNYDIP™—are a more far-reaching answer than the preceding ones. These circuits use the four extra pins to provide separate control inputs for *both* 8-bit internal groups, and also to provide improved functionality. The number of data pins is held at 16 by *multiplexing* the use of two 8-bit groups of input and/or output pins.

The motivation for 16-bit interface parts is, first of all, to cut component counts by replacing two parts with one in as many situations as possible, in order to save board space and assembly costs. Particularly in high-performance computers and array processors, the packaging itself is expensive when it must be designed to provide a proper signal-transmission environment for ultra-fast logic. An almost-50% cut in the board area required for the interface parts—here, as always, the "glue" which holds the whole system together—may result in major indirect savings.

But there are other incentives besides sheer cost reduction which favor cramming as much logic as possible into a given board area. There usually is only one board size in a chassis (or even in a system), and any logic subsystem which cannot fit onto one such board immediately incurs a *speed* penalty attributable to board-to-board communications—extra buffers for noise-free signal transmission, extra signal-path length on each board over to the edge where the connectors are, more extra length in the backplane wiring, and lots of additional inductance and capacitance permeating all of the above.

So, saving board area is very likely to improve *both* system cost *and* system performance, by increasing the probability that a given logic subsystem will fit onto just one board.

Interface-part internal element density has for many years been increasing at a rate which is, to say the least, unspectacular. Going from four to six to eight to sixteen logical elements in an interface-circuit package doesn't seem like a whole lot, compared for instance to going from 1K to 4K to 16K to 64K to 256K bits in a single dynamic-MOSRAM package in roughly the same number of years.

But, consider what a *true* LSI interface circuit would have to look like—one with the same magnitude of "equivalent gate count" being bandied about for today's microprocessors, dynamic MOSRAMs, and so forth. First of all, it would need to have *several hundred data inputs* and *several hundred data outputs,* so that the most immediately-plausible mechanical design for a package would resemble a sea urchin! And, if it were implemented using any present-day TTL technology, the part would dissipate enough *watts* to need cooling fins like a Porsche cylinder head!

And so it has turned out that progress over time in increasing the logical-element density for interface parts has been more or less linear, while progress in increasing the level of integration for microprocessors and dynamic MOSRAMs has been more or less exponential. It is no accident that a basic phrase of the definition for "interface circuits" quoted earlier in this paper is **"... which do not lend themslves to higher levels of integration ..."** If these same density trends continue, digital electronic systems of the future may actually have a *higher* proportion of packages allocated to interface circuits than is typical today, which if it happens is likely to surprise quite a few people.

Structure of 16-Bit Interface Circuits

Common configurations of two 8-bit interface parts used together furnish a natural starting point for the definition of useful 16-bit interface parts. When the same configuration tends to occur over and over again, it is natural to "draw a boundary around it and put it all on one chip," unless of course the resulting compound chip turns out to need too many pins.

Figure 5 illustrates three such two-part configurations which are observably very common, and intuitively very plausible:

- "Back-to-back" or "cross-coupled." (Figure 5A).
- "Nose-to-tail" or "pipelined." (Figure 5B.)
- "Side-by-side" or "parallel." (Figure 5C.)

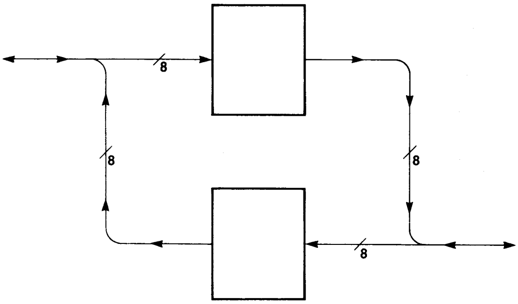


Figure 5A. Back-to-Back Configuration

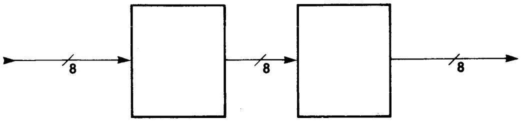


Figure 5B. Nose-to-Tail Configuration

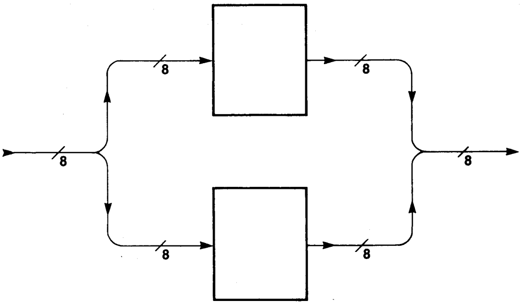


Figure 5C. Side-by-Side Configuration

Figure 5. Common Configurations of Two 8-Bit Interface Parts

The back-to-back configuration, when applied to simple 8-bit buffers, leads to buffer transceivers such as the 'LS245. The 'LS245 is, of course, still a 20-pin part; the choice was made to change its enable structure from that which would be strictly implied by placing two 'LS244s back-to-back, in order to hold the package size to 20 pins and to disallow having both directions simultaneously enabled. These same statements continue to hold for the 'LS645 and 'LS645-1. The 'LS640 and 'LS640-1 are inverting buffer transceivers, and the 'LS643 and 'LS643-1 incorporate an 8-bit inverting buffer back-to-back with an 8-bit noninverting buffer; there are also open-collector equivalents to these parts and the 'LS645 and 'LS645-1. The entire series features the same enable structure, with a master enable line $\overline{E}$ controlling *both* sets of buffers and a direction line DIR to allow just one direction to be enabled at a time.

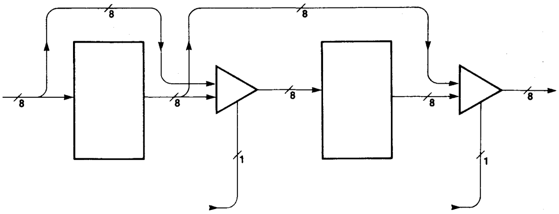


Figure 6. Two-Stage Pipeline Register Configuration

Applied to 'LS373 latches and 'LS374 registers, the back-to-back configuration leads to the 24-pin 'LS547 latch transceiver and the 'LS546 register transceiver respectively. These parts are just what one would expect them to be, with individual output-enable and clock control inputs for each 8-bit group, except that there are enough pins to also give each group clock-enable control inputs like the 'S377. The 'LS567 and 'LS566 are the corresponding inverting parts.

The nose-to-tail and side-by-side configurations do not lead to anything very interesting with buffers, at least as long as there are only enough pins for one 8-bit input data path and one 8-bit output data path. Latches and registers, however, are entirely another matter. It turns out to be attractive to *combine* these two configurations, even though at first glance they look quite dissimilar, into a single "two-stage pipeline" configuration as shown in Figure 6. Such a two-stage pipeline can operate in either a nose-to-tail mode or a side-by-side mode, according to the setting of the two internal multiplexers shown in Figure 6. Applied to 'LS373 latches and 'LS374 registers, this more powerful configuration leads to the 24-pin 'LS549 latch pipeline and the 'LS548 register pipeline. For these parts, the control inputs are a final-stage output enable, selects for each mux, a common clock (or latch-enable for the 'LS549) input for both stages, and individual clock-enable inputs for each stage.

To clarify the timing control of these parts, the 16-bit register parts ('LS546, 'LS566, and 'LS548) have individual clock-enable signals for each 8-bit group, and either individual clock signals ('LS546 and 'LS566) or a common clock signal ('LS548). The 16-bit latch parts ('LS547, 'LS567, and 'LS549), since the "clock" signal turns into a level-sensitive latch-enable signal, have two independent ways of enabling storage in each of the two stages. Thus, the 'LS547 and 'LS567 parts feature two separate and equivalent latch-enable control inputs for each 8-bit group, either one of which can cause the group to "latch up" and store information. The 'LS549 part has the same operating mode, except that each 8-bit group has one *separate* latch-enable control input and there is one more latch-enable input *common* to both groups.

As with other TTL 8-bit latches and registers, the part-numbering scheme assigns odd numbers to latches and even numbers to registers.

Front-loading latches are one other type of 16-bit interface part. The 'LS646 (noninverting) is to a first approximation an 'LS645 superimposed upon an 'LS546. (The numbering scheme wasn't *planned* to be that cute—it just happened.) The 'LS648 is a similar inverting part. To clarify what is meant, each of the eight logical elements of an 'LS646 consists of two back-to-back buffers and two back-to-back flipflops, with a parallelled buffer and flipflop pointing in the A-to-B direction and a similar buffer-flipflop pair pointing in the B-to-A direction. The 'LS646 and 'LS648 are three-state parts; there are also equivalent open-collector parts, and some other similar parts with a slightly different control structure.

32-bit interface parts are also visible on the horizon. Two four-stage pipelines, the Am29520 and Am29521, are offered by AMD as members of a series of signal-processing parts, and Monolithic Memories plans to make them also. As compared to the 'LS548 and 'LS549, they offer twice as many stored bits per square inch of board, but considerably less flexibility in accessing and controlling register contents.

The matrix approach to classifying various interface parts can be extended to encompass transceivers and pipelines, as is done in Table 4. The correspondence between the various 8-bit simple interface parts and the 16-bit compound interface parts which are in a sense derived from them, is summarized in Table 5.

Configuration	Buffers	Latches	Registers	Front-Loading Latches
Simple	'210 '310 '240 '340 '241 '341 '244 '344	'373 '531 '533 '535	'374 '532 '534 '536	---
Back-to-Back	'245 '640 '640-1 '643 '643-1 '645 '645-1	'547 '567	'546 '566	'646 '648
Two-Stage Pipeline	---	'549	'548	---

Table 4. Matrix Classification Scheme for 8-Bit and 16-Bit Interface Parts

Simple Interface Type	Compound Interface Type	Number Of Pins	Buffer	Latch	Register
	Transceivers:				
'244	'245 '645 '645-1	20	X		
'240	'640 '640-1	20	X		
'240/'244	'643 '643-1	20	X		
'373	'547	24		X	
'374	'546	24			X
'533	'567	24		X	
'534	'566	24			X
	Pipelines:				
'373	'549	24		X	
'374	'548	24			X

Table 5. Equivalences Between Simple and Compound Interface Types

Various Applications of Interface Parts

Some Logic-Design Examples

Several illustrative designs using various interface parts may suggest some design insights and some creative ways to use interface. The designs presented have generally been excerpted from actual digital systems.

Reading a switch setting to establish an externally-defined system parameter, such as a device address, is a mundane but essential task in many microprocessor-based systems. Figure 7 illustrates how a group of eight switches may conveniently be read using a byte-wide buffer such as the 'LS244. Since the switches must be electrically isolated from the bus, the 'LS244's three-state outputs are disabled by control signals originated by the microprocessor until the time comes to read in the switch settings. Because the 'LS244 can supply up to 24 milliamps of I_{OL} to drive the bus, this simple scheme can be utilized even on heavily-loaded system data buses.

If still more drive capability is needed, an 'S244 in the same configuration can sink up to 64 milliamps. And, if the system is to be operated in an industrial environment and the switch signals entering the buffer inputs are subject to severe noise, the Schmitt-trigger 'LS344 type of buffer can also be substituted for the 'LS244 with no other change to the circuit.

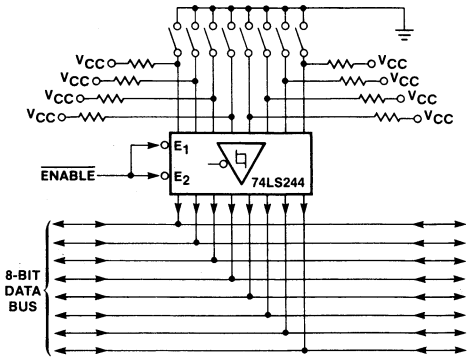


Figure 7. Switch-Setting Readin Circuit

Interfacing two separate buses is a very standard application for transceivers. Figure 8 shows an 'LS245, which has a control structure such that one control signal selects the *direction* of data transfer and the other one independently allows data transfer to be *enabled or disabled.* Thus, the two buses can be operated totally isolated from each other, or else either one may be made to follow the other. Depending on the drive-capability and polarity requirements of the application, any of the other buffer transceivers might be used here instead. Or, if memory as well as cross-coupling is required, a latch transceiver or register transceiver might also be used in a similar manner.

Driving a dynamic-MOSRAM address bus with a multiplexed row/column address can conveniently be done with an 'S700 as shown in Figure 9. This part is an inverting complementary-enable buffer with a series-resistor output structure, which is an ideal combination of characteristics here.

First of all, a TTL inverting buffer normally has one less transistor — and hence one less delay — in its internal data path than does an equivalent noninverting buffer, and hence is faster. And dynamic MOSRAMs really don't care if their addresses come in "true" or "complemented" form as long as that form *never* changes.

Second, a complementary-enable buffer can easily multiplex two different address sources to the same set of outputs without introducing extra switching delay, or allowing a momentary "bus fight" condition, if the same control signal (here CAS or "Column Address Strobe") is tied directly to both E_1 and E_2 and the two 4-bit groups of outputs are tied together.

Finally, because of the internal series resistor in the'S700's output structure, this part (like the 'S730/1/4) can drive highly capacitative loads, of say up to 70 dynamic-MOSRAM inputs, without the need for external limiting resistors to control undershoot, resulting in a net system speed gain since signal rising and falling transition times remain symmetric. Otherwise, the effective logic delay of the buffer (which is simply the worse of the two transition times) would get degraded, since the use of an external series resistor would have greatly lengthened the low-to-high transition time.

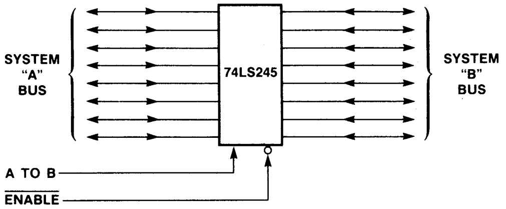


Figure 8. Interfacing Two Separate Buses

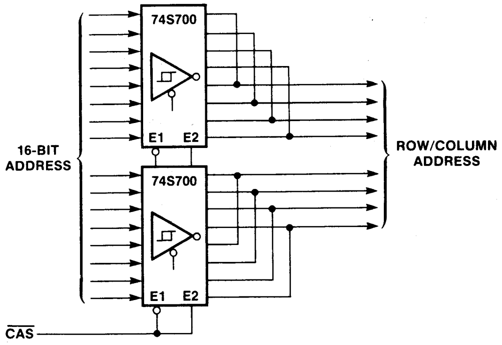


Figure 9. Multiplexed Row/Column Address Drivers

Demultiplexing and holding address and data words for single-bus microprocessors is an application which takes advantage of the strong points of the 'S531 as shown in Figure 10. Since the 'S531 is a "transparent latch" and can operate as a buffer when necessary, the memory system designer can take advantage of the *full* time slots when the address and data signals are present on the microprocessor outputs. Because the address and data signals are then present for a longer period of time at the 'S531 outputs, it may be possible to use slower (and therefore less expensive!) memory devices than if edge-triggered registers had been used here instead. The three-state outputs of the 'S531 allow the designer to implement bidirectional data

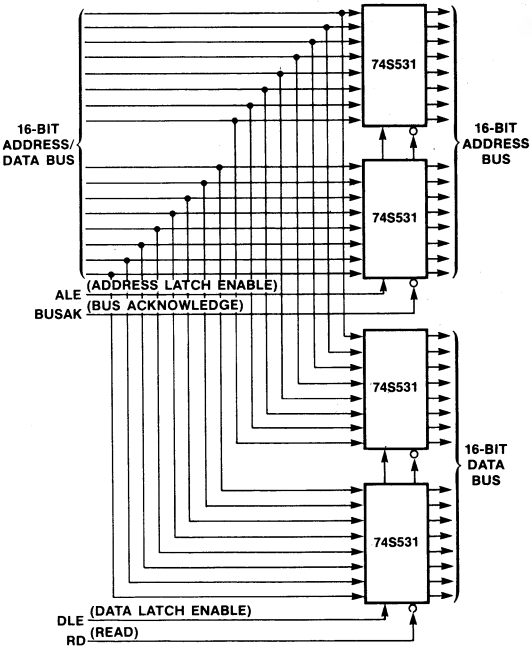


Figure 10. Address/Data Demultiplexer for Single-Bus Microprocessors

buses and DMA address schemes. Variations on this approach can use 'S373s if less drive capability is needed, or 'LS373s if less speed is needed as well; or 'S535s, 'S533s, or 'LS533s under the same respective circumstances if the address and data buses to be driven are assertive-low according to the system definition. If the data-bus interface needs to have latching capability also for data returning to the microprocessor, then 'LS547s are an excellent choice.

Synchronizing the state changes of a PROM-based control sequencer is easily performed using a register with a clock-enable feature, like the 'LS377 shown in Figure 11. In this simple sequencer, a 4-bit counter steps through the PROM addresses. The counter may be reset to address 0000, or loaded with any 4-bit address. The 32 × 8 PROM, with five address lines, allows for one external input as well as the four bits from the counter. The PROM outputs are pipelined using the 'LS377, which eliminates PROM output glitches, synchronizes the state changes of the sequencer with the system clock, and speeds up the effective cycle time. The availability of enable control inputs on both the counter and the 'LS377 allows forcing "wait" states, where both the counter and the register hold their current state for extended periods of time. If a higher-speed implementation of this design is needed, a 74S161 or 93S16 counter can replace the 74LS161, one of Monolithic Memories' new 63S081 ultra-speed 32 × 8 PROMs (25 nanoseconds worst-case and 9 nanoseconds typical for t_{AA}, instead of 50 and 37 nanoseconds respectively) can replace the 6331-1, and an 'S377 can replace the 'LS377.

Saving Designs at the Last Minute, or Planning Ahead

Designs hanging out over the edge of unworkability can sometimes be salvaged without any redesign effort, by replacing standard interface parts with hi-drive, Schmitt-trigger-input, or even just inverting pin-compatible parts. Hi-drive parts such as the 'S532 or 'LS645-1 get dropped into 'S374 or 'LS645 sockets respectively late in the design cycle, when the designer suddenly discovers that he has hung several too many inputs on his main system bus. Schmitt-trigger-input parts such as the 'LS341 likewise get dropped into 'LS241 sockets shortly after the designer has recovered from his first observation of his actual bus wave-forms on a good laboratory oscilloscope—it's that or back to the old drawing board. And, when he suddenly remembers after laying out a tightly packed board that "Oh, xxxx, that particular bus is assertive-*low,*" it's nice to be able to simply substitute an 'S534 for an 'S374 in a few places rather than having to find room for several inverter packages. So a designer who has learned to think of interface parts in terms of the matrix approach will now and then find a particularly quick route to saving his skin.

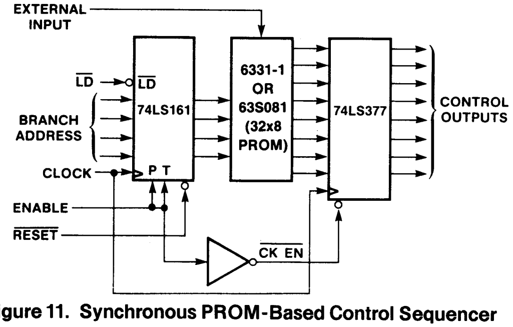


Figure 11. Synchronous PROM-Based Control Sequencer

However, an astute designer may use hi-drive, Schmitt-trigger-input, and inverting parts quite deliberately in order to gain speed, economy, drive capability, or noise immunity. A number of the industry-standard buses in the microcomputer world are assertive-low; and inverting buffers, latches, and registers are much more appropriate for connecting these to a microprocessor, or to a bit-slice arithmetic unit, than non-inverting parts with extra inverters in series just to make the polarity come out right. Similarly, Schmitt-trigger hex inverters whose only function in the data path is to provide noise immunity can be eliminated by using 'LS340-type buffers, which also provide significant drive capability and three-state outputs. The need to parallel three-state drivers and registers and split drive lines, just for extra drive capability, can be reduced or eliminated by using hi-drive parts. And, in an obvious but not trivial switch, substituting a high-speed Schottky part for a low-power Schottky equivalent part can beef up drive capability considerably.

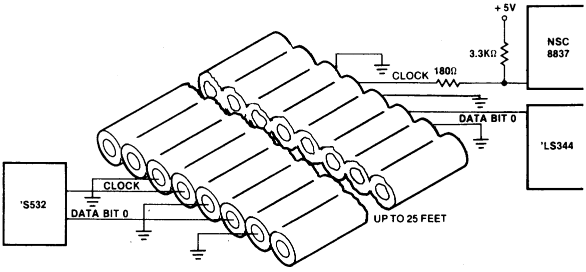
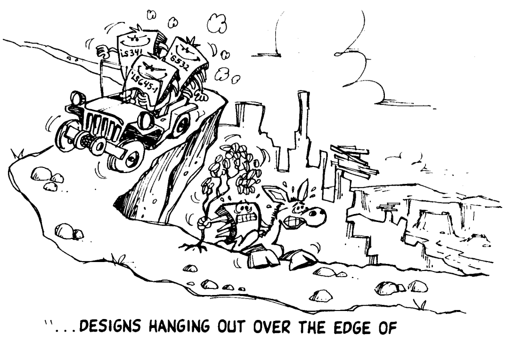


Figure 12. Flat-Cable Transmission Scheme Using Hi-Drive and Schmitt-Trigger-Input Interface Parts

Board-to-board signal transmission via flat cable is a particularly nice application for both hi-drive and Schmitt-trigger-input interface parts. The 32-milliamp outputs of, say, an 'S532 are better matched to the characteristic impedance of flat cable (usually 100 to 120 ohms) than 20-milliamp outputs would be. An adequate scheme, in many cases, for the transmission of *data* from board to board uses 3M or similar flat cable. Every second cable wire is grounded at both ends for shielding, so that signal wires alternate with ground wires ("signal-ground-signal-ground"), and there is at least one ground wire at each edge of the cable. Signal wires are driven by 32-mA hi-drive latches or registers, and the receivers are Schmitt-trigger-input buffers, and that's all there is to it—no resistors, capacitors, or black magic. For a strobe, clock, or control signal, a linear receiver such as a National Semiconductor 8837 is used together with a 180-ohm series resistor and a 3300-ohm shunt resistor to V_{CC}, as shown in Figure 12. This overall scheme is compatible with some Digital Equipment Corporation buses, and is good for transmission distances of up to 25 feet.



"...DESIGNS HANGING OUT OVER THE EDGE OF UNWORKABILITY CAN SOMETIMES BE SALVAGED..."

Conclusion

Interface parts seem primitive alongside of LSI microprocessors and dynamic MOSRAMs, but they are inescapable and smart designers today have learned how to use them astutely. A powerful aid in doing so is to think of the set of interface parts as an array, which fits into a matrix whose dimensions are various circuit properties. Even though the rate of progress seems slow, the bit-density and functionality of interface parts is steadily increasing, and the time is approaching for designers to learn to take the next logical step and use 16-bit interface parts extensively in their systems, in order both to save cost and to improve overall system performance.

Acknowledgements

The matrix approach to interface parts, as far as we know, originated with John Birkner (wh o also invented PALs®) and Zelimir Diel at Monolithic Memories. The "Aha!" insight of how to combine the nose-to-tail and side-by-side dual-latch and dual-register configurations into the two-stage pipeline configuration came from Josh Rosen at Computervision. The cartoons, which hopefully also help some particular points to stick, are the work of Marty Lindquist, who freelances with Designers Two in Palo Alto, CA. The flat-cable scheme we owe to John Zett of Melpar.

References

r1. "Bottom-Up Design with LSI and MSI Components," Chuck Hastings, *Conference Proceedings of the Fourth West Coast Computer Faire.* 11-13 May 1979, pages 359-365. Available from the Computer Faire, 333 Swett Road, Woodside, CA 94062.

r2. *Bipolar LSI 1982 Databook,* Monolithic Memories, Inc., 1165 East Arques Avenue, Sunnyvale, CA 94086. Interface parts are generally in section 12; however, the 'LS380 multifunction register is in section 8.

r3. "Architecture of the IBM System/360," G. M. Amdahl, G. A. Blaauw, and F. P. Brooks, *IBM Journal of Research and Development,* Volume 8 (1964), pages 87-101.

r4. "The 20-Pin Octal Interface Family—Today's Computer-System Building Blocks," Chuck Hastings, applications note available from Monolithic Memories, Inc. (see r2 for address) A longer paper written when buffer transceivers were the only visible 16-bit parts, but with more detail on the 8-bit parts.

Notes

Pseudo Random Number Generator (a disguised PAL)

By Nadia Sachs

"Due to their interesting properties, Pseudo Random Numbers (PRN) are useful across a wide spectrum of applications, including secure communication, test pattern generation, scramblers, and radar ranging systems. For the requirements of a given application, a "customized" PRN generator is readily implemented using PALs.

Pseudo Random Number Generator (a disguised PAL)

Considering a random succession of bits such as 1001011, one could hardly suspect the volumes of beautiful mathematics investigating such sequences, or the large amount of applications exploiting their interesting properties. Used for secure communication, polynomial counters, radar ranging systems, picture coding, waveform synchronization, spread spectrum modulation, scramblers, to name a few, random sequences are easily generated using PALs programmed as a feedback shift register.

Pseudo Random Sequences (PRS)

The random sequence 1001011 could very well be the recorded outcome of a coin tossed 7 times (heads = 1, tails = 0) in which case the sequence is referred to as random because of the circumstances of its generation; also it has some specific randomness properties. The same sequence obtained with digital logic (in a deterministic way) that satisfies randomness properties is termed "Pseudo-Random" (PR).

The randomness property that a PRS has to satisfy are:

1) The Balance Property
   The numbers of ones in the sequence differ from the numbers of zeroes by at most 1.
2) The Run Property
   Short run of consecutive zeroes or ones is more frequent than long runs. More precisely 1/2 the runs are of length one; 1/4 of length two; 1/8 of length three; and so forth.
3) The Autocorrelation Property
   If the sequence is compared term by term with any cyclic shift the number of agreements differs from the number of disagreements by at most 1.

Useful Properties of the PRS Sequences

To get a better insight as to why PR sequences are so useful we'll consider the significance of some of their properties.

The obvious property is their randomness which makes them a good candidate for Monte Carlo statistical techniques. Added to randomness their deterministic, reproducible generation makes PRS suitable as a "key" for encryption.

One of the most important properties is their autocorrelation. Consider the following experiment:

A PR sequence is repeatedly transmitted at point A: 1001011 1001011 1001011...At point B, the sequence is received and one desires to know the exact starting point; the original sequence 1001011 is compared with 7 bits from incoming sequence ie. with 1001011, 0010111, 0101110, etc. Note that the original is compared with a cyclic shift of itself. A measure of similarity of two sequences is the autocorrelation function R. A normalised definition of R is given by:

R = (Na-Nu)/sequence length

where Na = number of agreements
Nu = number of disagreements

R is plotted in Fig. 1 for all possible cyclic shifts. It consists of a narrow triangle around the zero delay and is essentially zero otherwise. Cyclic shifts of the sequence are uncorrelated or otherwise stated orthogonal.

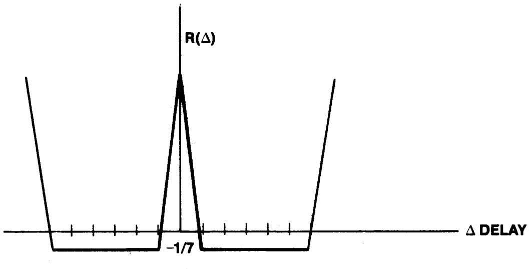


Figure 1. Autocorrelation of a PN Sequence 1001011

The meaning of this "two level" autocorrelation is that even corrupted by noise, the original sequence is readily distinguished from all other.

Taking advantage of this property are applications such as waveform sychronization, radar range determination, measurement of impulse response of a system, error detection schemes.

Due to the similarity of the autocorrelation of a PRS and White Gaussian Noise, the random sequences are also called "Pseudo Random Noise" (PRN). Noise behavior with bandlimited flat spectrum accounts for applications such as: spread spectrum and picture coding.

An important aspect that makes PR sequences useful is their ease of generation.

Generation of PRN Sequences

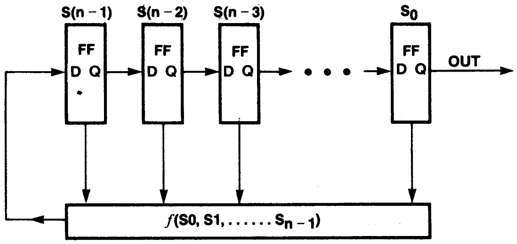


Figure 2. A n Stage Shift Register with Feedback

PRN sequences can be obtained in various ways. One possible implementation is illustrated in Fig. 2: a shift register with feedback. On each clock cycle the information is shifted one position to the right with the feedback term preventing the register from emptying.

Any preassigned binary sequence can be obtained at the output given a suitable choice of the number of stages in the shift register (n), the initial condition and the feedback function f (s0,s1,s2...s(n−1)).

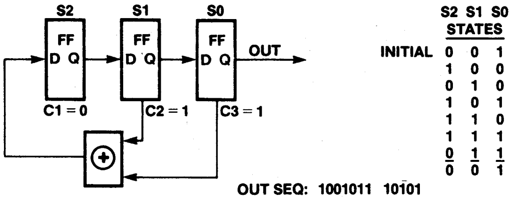
When the feedback function is expressed in the form:

$$f(s0,s1,s2 \ldots s(n-1)) = c_n * s0 \oplus c_{n-1} * s1 \oplus \ldots \oplus c_1 * s(n-1)$$

where each of the constants c is either 0 or 1 and ⊕ denotes addition modulo-2, the shift register is called linear. An example of a 3 stage linear shift register is shown in Fig. 3 for $c_1 = 0$ and $c_2 = c_3 = 1$.

The output sequence is ultimately periodic with a period p not exceeding 2^n. In the case of the linear shift register the period is at most 2^n-1; an output sequence achieving $p = 2^n-1$ is called a maximum length sequence.

For example, the 3 stage linear shift register in Fig. 3 generates the sequence 1001011 1001011...whose period is p = 7. In a maximum length sequence, all set of n consecutive bits except all zeroes occur. Note in Fig. 3 all states of a 3 bit binary counter (except 000) appear in a random order.



	S2	S1	S0
		STATES	
INITIAL	0	0	1
	1	0	0
	0	1	0
	1	0	1
	1	1	0
	1	1	1
	0	1	1
	0	0	1

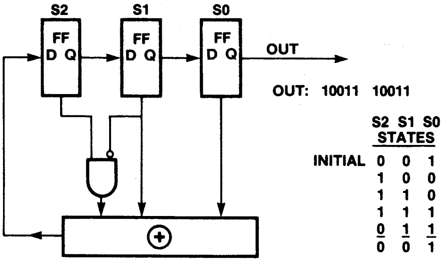
OUT SEQ: 1001011 10101

Figure 3. A 3 Stage Linear Shift Register with Feedback f (s0,s1,s2) = s0⊕ s1

Given a preassigned sequence, the feedback function can be determined via a truth table and minimization techniques. If no specific sequence is desired for an n stage register, there are many linear logics giving sequences of length 2^n-1. Table 1 gives an example of simple logic for generating sequences with various periods up to 1024K. Appendix 1 explains how these functions were obtained based on known results from polynomial arithmetic.

From a feedback shift register generating maximum length sequences, all period sequences from 1 to 2^n-2 could be obtained.

By adding the logic shown in Fig. 4 a sequence with period 5 is obtained using the original p = 7 sequence. The additional logic was determined by the method described in Appendix 2.



OUT: 10011 10011

	S2	S1	S0
		STATES	
INITIAL	0	0	1
	1	0	0
	1	1	0
	1	1	1
	0	1	1
	0	0	1

Figure 4. Generating a sequence with period 5. An AND gate is added to a maximum length linear 3 stage shift register with feedback.

PALs: Natural PRN Generators

PALs (16R8, 16R6, 16R4, 20R8, 20R6, 20R4, 20X10, 20X8, 20X4, 16X4, 16A4) are ideally suited to implement different size linear shift registers with feedback. As ten is the maximum number of stages in a single PAL, the maximum period attainable without cascading is 1023. Inspection of Table 1 reveals that up to n = 20 the feedback function will contain at most four terms. Four term modulo 2 addition is readily implemented with PAL as follows:

$$a \oplus b \oplus c \oplus d = (a \oplus b) \oplus (c \oplus d) \text{ associative law}$$
$$= (a{*}/b + /a{*}b) \oplus (c{*}/d + /c{*}d)$$

+ denotes OR function

This last form as shown in Fig. 5 corresponds to the fixed array of the ____ X ____ PAL family.

The additional logic to obtain various periods from a maximum length sequence can be included in the PAL.

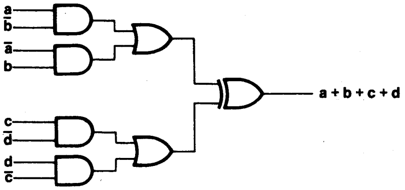


Figure 5. Four Term Modulo 2 Addition in PALs

Applications

Due to their properties and ease of generation, PRN sequences are widely used. Some of their applications will be briefly covered.

Polynomial Counters

The n parallel output of a maximum length linear shift register cycles through all the states of a n bit binary counter (except zeroes) in a random order.

When used to replace binary counters for sequencers or "count down" applications, the feedback shift register is referred to as a polynomial counter. The example in Fig. 3 is a 3 bit polynomial counter.

Polynomial Counters are advantageous because they are usually smaller (fewer transistors needed) and faster as there is just a shift no ripple carry.

Private Communication

A basic diagram for encipherment is shown in Fig. 6.

A "key," the output of a PRN generator, is added modulo 2 to the message. At the receiver the "key" is subtracted which is equivalent to addition modulo 2.

The key is random and by synchronizing the two identical shift registers of the transmitter and the receiver the need for key transmission is eliminated.

More sophisticated systems based on this simple Vigenere cipher can be conceived.

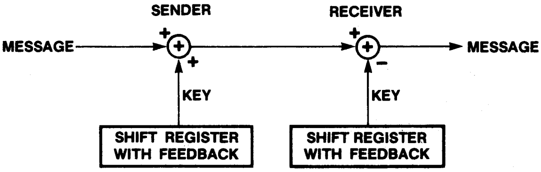


Figure 6. Secure Communication System Using Maximum Length Sequences.

Picture coding using PRN quantizer

The standard procedure for digital coding is to uniformly quantize the luminance signal; ie., divide the continuous interval into equal sections and assign a code to each section. This process produces a picture which has visible steps of intensity unless a minimum 6 bits per sample is used.

The contour effects can be suppressed by PRN quantization, taking advantage of subjective aspects of vision. The human observer is much more annoyed by tonal errors, as that produced by a 3-4 bit quantizer, than by noise because of the eye's ability to average out noise.

By adding noise (dither) a particular intensity can result in different outputs at different times; the average intensity of an area is reproduced correctly, more noisily, but that is less objectionable than the step-shaped effect.

The pseudo random source should not repeat for several frames and usually the noise should affect the least significant bit of the quantizer. Reasonable image quality is achieved using PRN quantization with 3-4 bits per sample and 16 stages PRN generators.

Appendix 1

Generation of a maximum length sequence.

The method of analysis is outlined and some results are summarized to help understand the derivation of Table 1. For a detailed and complete presentation see Ref. 1.

The linear shift register can be described in terms of the initial condition: $\{a_{-1}\ a_{-2}\ a_{-3} \ldots a_{-n}\}$ and the feedback function. Any stage of the register can be expressed as:

$$a_m = \sum_{i=1}^{n} c_i\, a_{m-i} \qquad \text{(recurrence relation)}$$

Over a period of time any stage of the register will go through a sequence of states $\{a_m\} = \{a_0, a_1, a_2 \ldots\}$

This sequence can be associated with a generating polynomial

$$G(d) = \sum_{m=0}^{\infty} a_m\, d^m$$

where d can be thought of as a delay element and each state occurs after a set delay in time.

Replacing a_m with the recurrence equation in the generating polynomial and, after some algebra (see Ref. 1), one obtains:

$$G(d) = \frac{\sum_{i=1}^{n} c_i d^i (a_{-i} d^{-i} + a_{-1} d^{-1})}{1 - \sum_{i=1}^{n} c_i d^i}$$

G(d) is a function of the initial condition and the feedback coefficients c_i.

The denominator $D(d) = 1 - \sum_{i=1}^{n} c_i d^i$ will be referred to as the characteristic polynomial.

It can be further proved that the period p of the sequence generated is the smallest positive integer p for which D(d) divides $1-d^p$When D(d) is irreducible, the period of the shift register does not depend on the initial condition, except for the initial condition "all 0's."

D(d) irreducible is also a necessary, but not sufficient, condition for the sequence to be of maximum length.

From the class of irreducible polynomials an appropriate D(d) is obtained. Once D(d) is determined the shift register is defined.

Appendix 2

Generation of sequences with period $p <= 2^n-2$

In the sequence $\{\alpha_m\}$ of period 2^n-1 there are two n symbol subsequences spaced p positions apart, which agree in the first n−1 positions but disagree in the n−th position.

In the example sequence with period $2^n-1 = 7$ (Fig. 3) the first 14 terms are:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}
1	0	0	1	0	1	1	1	0	0	1	0	1	1

The two subsequences spaced p = 5 position apart are $\{a_4, a_5, a_6\}$ and $\{a_9, a_{10}, a_{11}\}$; they agree in the first two terms and disagree in the last term.

This property is a consequence of the DELAY AND ADD property which states that: if a sequence is added term by term modulo-2 to a cyclic shift of itself the resulting sequence is another cyclic shift of the original sequence. The resulting sequence contains all sets of n consecutive bits except "all 0." It contains the subsequence 000...1 which means that the two terms added agreed on n−1 consecutive positions. For our example:

```
1001011   one period original seq
1110010   5 bit cyclic shift (left rotation)
0111001
```

The same two subsequences .011 and .010 are obtained.

If the pattern is identified as soon as present in the register, the similarity could be forced and the period limited to 5.

The detecting logic is /s1 * s2 and f(s0, s1, s2) will be modified to:

s0 ⊕ s1 ⊕ (/s1*s2)

This example is illustrated in Fig. 4.

n	Logic	(period 2^n-1)
0		S0 ⊕ S1
3		S0 ⊕ S1
4		S0 ⊕ S1
5		S0 ⊕ S2
6		S0 ⊕ S1
7		S0 ⊕ S1
8	S6 ⊕ S5 ⊕ S4 ⊕ S0	
9	S4 ⊕ S0	
10	S3 ⊕ S0	
11	S2 ⊕ S0	
12	S10 ⊕ S2 ⊕ S1 ⊕ S0	
13	S12 ⊕ S2 ⊕ S1 ⊕ S0	
14	S12 ⊕ S2 ⊕ S1 ⊕ S0	
15	S1 ⊕ S0	
16	S5 ⊕ S3 ⊕ S2 ⊕ S0	
17	S3 ⊕ S0	
18	S7 ⊕ S0	
19	S5 ⊕ S2 ⊕ S1 ⊕ S0	
20	S3 ⊕ S0	

TABLE 1

Reference

1. Golomb, S.W., "Shift Register Sequences," Holden-Day, Inc. 1967
2. Golomb, S.W., "Digital Communication with Space Application," Prentice-Hall, Inc. 1964
3. Roberts, L.G., "Picture Coding Using Pseudo Random Noise," IRE Trans. Inf. Theory, Vol. IT-8, pp 145-154, Feb. 1962
4. PAL Programmable Array Logic Handbook

Monolithic Memories World-Wide Applications Support

U.S.A.

California	
Brea	
Mike Vogel	(714) 556-1216
Canoga Park	
Michael Sholklapper	(714) 556-1216
San Jose	
Dan Jesswein	(408) 249-7766
Colorado	
Wheatridge	
Roger Rios	(303) 420-6093
Illinois	
Worth	
Bill Karkula	(312) 961-9200
Massachusetts	
Framingham	
Mike Volpigno	(617) 875-7373
Dick Kinsella	(617) 875-7373
Texas	
Scott Skillman	(214) 690-3812

EUROPE

England	
Joe Gabris	(252) 517431
Chris Jay	(252) 517431
France	
Jose Juntas	(1) 6874500
Michell Rolland	(1) 6874500
Germany	
Willy Voldan	(89) 984961
Peter Wittfoth	(89) 984961
Peter Zecherle	(89) 984961

JAPAN

Tokyo	
Sadahiro Horiko	(427) 29-3401
Mitsunori Sugai	(427) 29-3401

Monolithic Memories Franchised Distributors

U.S.A.

Alabama

Huntsville	
Hall-Mark Electronics	(205) 837-8700

Arizona

Phoenix	
Kierulff Electronics	(602) 243-4101
Tempe	
Anthem Electronics	(602) 244-0900
Bell Industries	(602) 966-7800
Marshall Electronics Group	(602) 968-6181

California

Canoga Park	
Marshall Electronics Group	(213) 999-5001
Chatsworth	
Anthem Electronics	(213) 700-1000
Arrow Electronics	(213) 701-7500
El Monte	
Marshall Electronics Group	(213) 686-0141
Irvine	
Marshall Electronics Group	(714) 556-6400
Los Angeles	
Kierulff Electronics	(213) 725-0325
Newport Beach	
Arrow Electronics	(714) 851-8961
Palo Alto	
Kierulff Electronics	(415) 968-6292
San Diego	
Anthem Electronics	(619) 279-5200
Arrow Electronics	(619) 565-4800
Kierulff Electronics	(619) 278-2112
San Jose	
Anthem Electronics	(408) 946-8000
Sunnyvale	
Arrow Electronics	(408) 745-6600
Marshall Electronics Group	(408) 732-1100
Tustin	
Anthem Electronics	(714) 730-8000
Image Electronics	(714) 730-0303
Kierulff Electronics	(714) 731-5711

Colorado

Aurora	
Arrow Electronics	(303) 696-1111
Englewood	
Anthem Electronics	(303) 790-4500
Kierulff Electronics	(303) 790-4444
Wheatridge	
Bell Industries	(303) 424-1985

Connecticut

Wallingford	
Arrow Electronics	(203) 265-7741
Kierulff Electronics	(203) 265-1115
Marshall Electronics Group	(203) 265-3822

Florida

Clearwater	
Hall-Mark Electronics	(800) 282-9350
Fort Lauderdale	
Arrow Electronics	(305) 776-7790
Hall-Mark Electronics	(305) 971-9280
Kierulff Electronics	(305) 486-4004
Orlando	
Hall-Mark Electronics	(305) 855-4020
Palm Bay	
Arrow Electronics	(305) 725-1480
St. Petersburg	
Kierulff Electronics	(813) 576-1966

Georgia

Norcross	
Arrow Electronics	(404) 449-8252
Hall-Mark Electronics	(404) 447-8000
Kierulff Electronics	(404) 447-5252

Illinois

Bensenville	
Hall-Mark Electronics	(312) 860-3800
Elk Grove Village	
Kierulff Electronics	(312) 640-0200
Schaumburg	
Arrow Electronics	(312) 397-3440

Indiana

Indianapolis	
Advent Electronics	(317) 872-4910
Arrow Electronics	(317) 243-9353

Iowa

Cedar Rapids	
Advent Electronics	(319) 363-0221

Kansas

Lenexa	
Hall-Mark Electronics	(913) 888-4747

Maryland

Baltimore	
Arrow Electronics	(301) 247-5200
Columbia	
Hall-Mark Electronics	(301) 988-9800
Gaithersburg	
Pioneer Washington	(301) 948-0710
Linthicum	
Kierulff Electronics	(301) 247-5020

Massachusetts

Billerica	
Kierulff Electronics	(617) 667-8331
Burlington	
Lionex	(617) 272-9400
Marshall Electronics Group	(617) 272-8200
Woburn	
Arrow Electronics	(617) 933-8130

Michigan

Ann Arbor	
Arrow Electronics	(313) 971-8220
Grand Rapids	
RS Electronics	(616) 241-3483
Kalamazoo	
RS Electronics	(616) 381-5470
Livonia	
RS Electronics	(313) 525-1155

Minnesota

Bloomington	
Hall-Mark Electronics	(612) 941-7500
Edina	
Arrow Electronics	(612) 830-1800
Kierulff Electronics	(612) 835-4388

Missouri

Maryland Heights	
Hall-Mark Electronics	(314) 291-5350
St. Louis	
Arrow Electronics	(314) 567-6888

New Hampshire

Manchester	
Arrow Electronics	(603) 668-6968

New Jersey

Cherry Hill	
Hall-Mark Electronics	(609) 424-0800
Fairfield	
Arrow Electronics	(201) 575-5300
Kierulff Electronics	(201) 575-6750
Lionex	(201) 227-7960
Mt. Laurel	
Marshall Electronics Group	(215) 627-1920
Marlton	
Arrow Electronics	(609) 235-1900
Nutley	
Vantage Electronics	(201) 667-1840

New Mexico

Albuquerque	
Arrow Electronics	(505) 243-4566
Bell Industries	(505) 292-2700

New York

Buffalo	
Summit Distributors	(716) 884-3450
E. Syracuse	
Add Electronics	(315) 437-0300
Endwell	
Marshall Electronics Group	(607) 754-1570
Hauppauge	
Arrow Electronics	(516) 231-1000
Current Components	(516) 273-2600
Lionex	(516) 273-1660
Liverpool	
Arrow Electronics	(315) 652-1000
Melville	
Arrow Electronics	(516) 694-6800
Rochester	
Arrow Electronics	(716) 275-0300
Marshall Electronics Group	(716) 235-7620

North Carolina

Raleigh	Arrow Electronics	(919) 725-8711
	Hall-Mark Electronics	(919) 872-0712
	Resco Raleigh	(919) 781-5700

Ohio

Centerville	Arrow Electronics	(513) 435-5563
Cleveland	Kierulff Electronics	(216) 587-6558
Dayton	Marshall Electronics Group	(513) 236-8088
Solon	Arrow Electronics	(216) 248-3990
	Hall-Mark Electronics	(216) 349-4632
Westerville	Hall-Mark Electronics	(614) 891-4555

Oklahoma

Tulsa	Hall-Mark Electronics	(918) 835-8458
	Quality Components	(918) 664-8812
	Radio, Inc.	(918) 587-9123

Oregon

Beaverton	Almac Electronics	(503) 641-9096
Lake Oswego	Anthem Electronics	(503) 684-2661

Pennsylvania

Horsham	Pioneer/Delaware Valley	(215) 674-4000
Monroeville	Arrow Electronics	(412) 856-7000

Texas

Addison	Quality Components	(214) 387-4949
Austin	Hall-Mark Electronics	(512) 258-8848
	Quality Components	(512) 835-0220
Dallas	Arrow Electronics	(214) 386-7500
	Hall-Mark Electronics	(214) 234-7300
	Kierulff Electronics	(214) 340-8880
Houston	Arrow Electronics	(713) 530-4700
	Hall-Mark Electronics	(713) 781-6100
Sugarland	Quality Components	(713) 491-2255

Utah

Salt Lake City	Bell Industries	(801) 972-6969
	Kierulff Electronics	(801) 973-6913

Washington

Bellevue	Almac Electronics Corporation	(206) 643-9992
	Arrow Electronics	(206) 643-4800
Redmond	Anthem Electronics	(206) 881-0850
Tukwila	Kierulff Electronics	(206) 575-4420

Wisconsin

Oak Creek	Arrow Electronics	(414) 764-6600
	Hall-Mark Electronics	(414) 761-3000
Waukesha	Kierulff Electronics	(414) 784-8160

CANADA

Alberta

Calgary	Zentronics Limited	(403) 230-1422

British Columbia

Richmond	Zentronics Limited	(604) 273-5575
Vancouver	RAE Electronics	(604) 291-8866

Manitoba

Winnipeg	Zentronics Limited	(204) 775-8661

Ontario

Brampton	Zentronics Limited	(416) 451-9600
Nepean	Zentronics Limited	(613) 226-8840
Toronto	Future Electronics	(416) 663-5563
Waterloo	Zentronics Limited	(519) 884-5700

Quebec

Montreal	Future Electronics	(514) 694-7710
	Prelco Electronics	(514) 389-8051
St. Laurent	Zentronics Limited	(514) 735-5361

Monolithic Memories Representatives

U.S.A.

Alabama	**Huntsville**	REP, Inc.	(205) 881-9270
Arizona	**Scottsdale**	Summit Sales	(602) 998-4850
California	**Cupertino**	Thresum Associates	(408) 996-9889
	Fountain Valley	Bager Electronics	(714) 957-3367
	San Diego	Littlefield & Smith	(619) 455-0055
Colorado	**Wheatridge**	Waugaman Assoc.	(303) 423-1020
Connecticut	**North Haven**	Comp Rep Associates	(203) 239-9762
Florida	**Altamonte Springs**	Dyne-A-Mark	(305) 831-2097
	Clearwater	Dyne-A-Mark	(813) 441-4702
	Fort Lauderdale	Dyne-A-Mark	(305) 771-6501
	Palm Bay	Dyne-A-Mark	(305) 727-0192
Georgia	**Tucker**	REP, Inc.	(404) 938-4358
Illinois	**Rolling Meadows**	Sumer	(312) 991-8500
Indiana	**Indianapolis**	Leslie M. DeVoe Co.	(317) 842-3245
Iowa	**Cedar Rapids**	S & O Sales	(319) 393-1845
Kansas	**Olathe**	Rush and West	(913) 764-2700
Maryland	**Baltimore**	Monolithic Sales	(301) 296-2444
Massachusetts	**Westwood**	Comp Rep Associates	(617) 329-3454
Michigan	**Grosse Point**	Greiner Associates	(313) 499-0188
Minnesota	**Edina**	Technical Sales, Inc.	(612) 941-9790
Missouri	**Ballwin**	Rush and West	(314) 394-7271
New Jersey	**Teaneck**	R. T. Reid Associates	(201) 692-0200
New Mexico	**Albuquerque**	BFA Corporation	(505) 292-1212
New York	**East Rochester**	Tri-Tech Electronics, Incorporated	(716) 385-6500
	Endwell	Tri-Tech Electronics, Incorporated	(607) 754-1094
	Fayetteville	Tri-Tech Electronics, Incorporated	(315) 446-2881
	Fishkill	Tri-Tech Electronics, Incorporated	(914) 897-5611
North Carolina	**Charlotte**	REP, Inc.	(704) 563-5554
	Raleigh	REP, Inc.	(919) 851-3007
Ohio	**Cincinnati**	Makin Associates	(513) 871-2424
	Columbus	Makin Associates	(614) 459-2423
	Kent	Makin Associates	(216) 921-0080
	Mentor	Makin Associates	(216) 257-2961
Oklahoma	**Tulsa**	West Associates	(918) 492-0390
Oregon	**Portland**	Northwest Marketing	(503) 297-2581
Pennsylvania	**Glenside**	CMS Marketing	(215) 885-5106
Puerto Rico	**Mayaguez**	Comp Rep Associates	(809) 832-9529
Tennessee	**Jefferson City**	REP, Inc.	(615) 475-4105
Texas	**Austin**	West Associates	(512) 454-3681
	Dallas	West Associates	(214) 248-7060
	Houston	West Associates	(713) 777-4108
Utah	**Salt Lake City**	Waugaman Assoc.	(801) 261-0802
Washington	**Bellevue**	Northwest Marketing	(206) 455-5846
Wisconsin	**Brookfield**	Sumer	(414) 784-6641

CANADA

Ontario	**Brampton**	Cantec	(416) 791-5922
	Ottawa	Cantec	(613) 725-3704
	Waterloo	Cantec	(519) 744-6341
Quebec	**Dollard Des Ormeaux**	Cantec	(514) 683-6131

AUSTRIA

Ing. Ernst Steiner
Hummelgasse 14
A 1130 Wien
Phone: 22-8274740
Telex: 135026

AUSTRALIA

R & D Electronics Pty Ltd.
4 Florence St.
Burwood, Vic. 3125
Phone: 3-288-8911
Telex: AA33288

R & D Electronics Pty Ltd.
133 Alexander St.
Crows Nest—2065
Phone: 2-439-5488
Telex: AA25468

BELGIUM

D & D Electronics
7E Olympiadelaan 93
2020 Antwerp
Phone: 3-8277934
Telex: 73121

DENMARK

C-88 APS
Kokkedal Industripark 42A
DK-2980 Kokkedal
Phone: 244888
Telex: 41198

ENGLAND

Monolithic Memories Ltd.
Lynwood House
1 Camp Road
Farnborough
Hampshire
GU14 6EN
Phone: 9-011-44-25-517431
Telex: 858051 MONO UK G

Memory Devices Ltd.
5th Floor Hagley House
Hagley Road
Edgbaston
Birmingham B16 8QG
Phone: 021-455-9395
Telex: 339752 Analog G

Memory Devices Ltd.
Central Avenue
East Molesey
KT8 OSN
Phone: 1-9411066
Telex: 929962

Macro Marketing Ltd.
396 Bath Road
Slough, Berkshire
Phone: 628663011
Telex: 847083

Microlog Ltd.
First Floor, Elizabeth House
Duke Street
Woking, Surrey GU21 5BA
Phone: (04862) 66771
Telex: 859219 (ULOG G)

FINLAND

Insrumentarium Oy Elektronikka
P.O. Box 64 Vitikka 1
02631 ESP00 63
Finland
Phone: 9-011-358-5281
Telex: 124426 HAVUL SF

FRANCE

Monolithic Memories France S.A.R.L.
Silic 463
94613 Rungis Cedex
Phone: 1-6874500
Telex: 202146
FAX: 6860818

Bellion Electronique
Z.I. Kerscao/Brest
B.P. 16-29219 Le Relecq-Kerhuon
Phone: 9-(98)-28-03-03
Telex: 940930

Composants S.A.
Avenue Gustave Eiffel
B.P. 81-33605 Pessac Cedex
Phone: 9-(56)-36-40-40
Telex: 550696

Datadis S.A.
10-12 Rue Emile Landrin
92100 Boulogne
Phone: 9-1-6056000
Telex: 20195

Dimel
Le Marino
Ave. Claude Farrere
83000 Toulon
Phone: 94/414963
Telex: 490093

Generim
24 Avenue De la Hoville Blanche
B.P. 1-38170 Seyssinet
Phone: 1-(76)-49-14-49
Telex: 320000

Generim S.A.R.L.
Zone d'Activities de Courtaboeuf
Avenue de la Baltique
P.O. Box 88
91943 Les Ulis Cedex
Phone: 1-9077878
Telex: 691700

Jermyn
Immeuble Orix
16, Av. Jean Jaures
94600 Choisy Le Roi
Phone: 853-12-00
Telex: 260967

GERMANY

Monolithic Memories, GmbH
Mauerkircherstr 4
8000 Munich 80
Phone: 89-984961
Telex: 524385
Fax: 89-983162

Astronic GmbH
Winzerstrasse 47D
8000 Munich 40
Phone: 309011
Telex: 5216187

Dr. Dohrenberg Vertriess GmbH
Bayreuther Strabe 3
1000 Berlin 30
Phone: 030-2138043
Telex: 0184860

Electronic 2000 Vertriebs GmbH
Stahlgruberring 16
8000 Munich 82
Phone: 89-420010
Telex: 522561

Nordelektronik GmbH KG
Karl-Zeiss Str 6
2085 Quickborn
Phone: 04106-72072
Telex: 214299

Positron Bauelemente Vertriebs GmbH
Benzstrasse 1
Postfach 1140
7016 Gerlingen-Stuttgart
Phone: 07151/3560
Telex: 7245266

SES Electronics GmbH
Oettingerstrasse 6
8860 Nordlingen
Phone: 09081/6001

HONG KONG

CET Ltd.
1402-203 Hennessy Road
Wanchai, Hong Kong
Phone: 5-729376
Telex: 85148 CET HX

INDIA

Kryonix
Kowdiar
Trivandrum
PIN 695 003
Kerala India
Phone: 63805
Telex: 884-307

Micro Aids International
790 Lucerne Dr.
Suite 40
Sunnyvale, CA 94086

ISRAEL

Telsys Ltd.
12 Kehilat Venetsia St.
Tel Aviv
Phone: (3)494891-5
Telex: 032392

ITALY

Comprel S.R.L.
Viale Fulvio Testi 115
20092 Cinisello Balsamo/Milano
Phone: 2-6120641
Telex: 332484

JAPAN

Monolithic Memories Japan KK
5-17-9 Shinjuku
Shinjuku-Ku
Tokyo 160
Phone: 3-207-3131
Telex: 232-3390 MMI KK J
Fax: 3-207-3139

Comtecs Co., Ltd.
2-19-7 Higashi-Gotanda
Shinagawa-Ku
Tokyo 141
Phone: (03) 441-7100
Telex: 242-3509 CTSLEXJ

Internix Inc.
Shinjuku Hamada Bldg.
7-4-7 Nishi-Shinjuku
Shinjuku-Ku
Tokyo 160
Phone: (03) 369-1101
Telex: J26733

K. Tokiwa & Co.
Asahi-Seimei-Omori Bldg.
1-1-10 Omori-Kita
Shinagawa-Ku
Tokyo 143
Phone: (03) 766-6701
Telex: 246-6821
Fax: (03) 766-1300

Synderdyne Inc.
Ishibashi Bldg.
1-20-2 Dogenzaka
Shibuya-Ku
Tokyo 150
Phone: (03) 461-9311
Telex: J32457

KOREA

Duck Woo Trading Company
K.P.O. Box 570
Seoul, Korea
Phone: 725-1330
Telex: MOCNDM K23231

NETHERLANDS

Alcom Electronics B.V.
P.O. Box 358
2900 AJ Capelle
A/D Ijssel Holland
Phone: 31-10-519533
Telex: 26160

NORWAY

Henaco A/S
P.O. Box 126 Kaldbakken
Trondheimsveien 436 Ammerud
Oslo 9
Phone: 2-162110
Telex: 76716

SINGAPORE

Dynamar International Ltd.
Unit 05-11,
12 Lo Rong Bakar Batu
Kolam Ayer Industrial Estate
Singapore 1334
Phone: 7476188
Telex: RS26283

SOUTH AFRICA

Promilect Pty Ltd.
P.O. Box 56310
Pinegowrie 2123
Phone: 789-1400
Telex: 424822

SOUTH AMERICA

Intectra
2629 Terminal Blvd
Mountain View, CA 94043
Phone: (415) 967-8818
Telex: 345-545 INECTRA MNTV

Intectra Do Brasil
Av. Paulista 807-S/415
Sao Paulo
Phone: 285-6305
Telex: 01139872 BRCOBR

SPAIN

Sagitron
C/Castello, 25, 2
Madrid 1
Phone: (1)4026085
Telex: 43819

SWEDEN

Naxab
Box 4115
S 17104 Solna
Phone: 8-985140
Telex: 17912

SWITZERLAND

Industrade AG
Gemsenstrasse 2
CH 8021 Zurich
Phone: 01-3632230
Telex: 56788

TAIWAN

Multitech International Corp.
315 Fushing N. Road
Taipei 104, Taiwan R.O.C.
Phone: (2) 7134022
Telex: 23756 or 19162 MULTIIC

Multitech Electronics Inc.
195 W. El Camino
Sunnyvale, CA 94086
Phone: (408) 733-8400
Telex: 352070

Notes